US009451250B2

(12) **United States Patent**
MacInnis et al.

(10) **Patent No.: US 9,451,250 B2**
(45) **Date of Patent: Sep. 20, 2016**

(54) **BOUNDED RATE COMPRESSION WITH RATE CONTROL FOR SLICES**

(71) Applicant: **Broadcom Corporation**, Irvine, CA (US)

(72) Inventors: **Alexander Garland MacInnis**, Los Altos, CA (US); **Frederick George Walls**, Grafton, WI (US)

(73) Assignee: **Broadcom Corporation**, Irvine, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 336 days.

(21) Appl. No.: **14/044,612**

(22) Filed: **Oct. 2, 2013**

(65) **Prior Publication Data**

US 2014/0092960 A1 Apr. 3, 2014

Related U.S. Application Data

(60) Provisional application No. 61/709,316, filed on Oct. 3, 2012, provisional application No. 61/764,807, filed on Feb. 14, 2013, provisional application No. 61/764,891, filed on Feb. 14, 2013, provisional application No. 61/770,979, filed on Feb. 28, 2013, provisional application No. 61/810,126, filed on Apr. 9, 2013, provisional application No. 61/820,967, filed on May 8, 2013, provisional application No. 61/856,302, filed on Jul. 19, 2013, provisional application No. 61/832,547, filed on Jun. 7, 2013.

(51) **Int. Cl.**

H04N 7/12	(2006.01)
H04N 11/02	(2006.01)
H04N 11/04	(2006.01)
H04N 19/152	(2014.01)
H04N 19/149	(2014.01)
H04N 19/124	(2014.01)
H04N 19/174	(2014.01)

(52) **U.S. Cl.**

CPC ***H04N 19/00193*** (2013.01); ***H04N 19/124*** (2014.11); ***H04N 19/149*** (2014.11); ***H04N 19/174*** (2014.11)

(58) **Field of Classification Search**

CPC H04N 19/00193; H04N 19/174; H04N 19/149; H04N 19/124; H04N 19/152

USPC 375/240.03

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,650,860 A 7/1997 Uz
8,139,882 B2 3/2012 Huguenel et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0434427 A2 12/1990
EP 0495490 A2 7/1992

(Continued)

OTHER PUBLICATIONS

International Search Report, App. No. PCT/US2013/063232 dated Jan. 30, 2014, 6 pages.

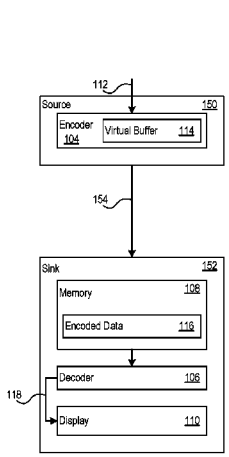
(Continued)

Primary Examiner — Mehrdad Dastouri
Assistant Examiner — Jared Walker
(74) *Attorney, Agent, or Firm* — Brinks Gilson & Lione

(57) **ABSTRACT**

A system implements rate control for encoding and decoding operations, for example, operations performed on slices of data such as image data. The system implements a transformation from actual buffer fullness to rate controlled fullness. With the rate controlled fullness model, the encoders and decoders adapt bit allocation responsive to rate controlled fullness, instead of the actual fullness.

20 Claims, 32 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0008781	A1	1/2004	Porter et al.	
2005/0015259	A1	1/2005	Thumpudi et al.	
2005/0069039	A1	3/2005	Crinon	
2005/0169370	A1	8/2005	Lee	
2005/0254577	A1*	11/2005	Ando	H04N 19/176 375/240.03
2006/0227870	A1	10/2006	Tian et al.	
2006/0268012	A1	11/2006	MacInnis et al.	
2007/0009163	A1	1/2007	Sasaki et al.	
2007/0071094	A1*	3/2007	Takeda	H04N 19/197 375/240.04
2007/0263720	A1*	11/2007	He	H04N 19/124 375/240.03
2009/0135921	A1	5/2009	Lei	
2009/0310672	A1*	12/2009	Rao	H04N 7/148 375/240.03
2010/0150227	A1	6/2010	Lee	
2010/0232497	A1*	9/2010	MacInnis	H04N 19/61 375/240.03
2011/0255595	A1	10/2011	Zuo	
2011/0274160	A1*	11/2011	Matsui	H04N 19/176 375/240.03
2011/0310967	A1	12/2011	Zhang	
2013/0051457	A1*	2/2013	Joshi	H04N 19/147 375/240.03
2013/0176431	A1	7/2013	Ogawa et al.	

FOREIGN PATENT DOCUMENTS

EP	0734174		9/1996
EP	2034741	A1	8/2006
GB	2346282	A	8/2000
WO	WO2010/091503	A1	8/2010
WO	WO2011/043793	A1	4/2011
WO	WO 2012/029208	A1	3/2012

OTHER PUBLICATIONS

International Search Report, App. No. PCT/US2013/063237 dated Jan. 30, 2014, 6 pages.

Malvar, Henrique S., et al., Lifting-based reversible color transformations for image compression, SPIE, vol. 7073, 2008, 10 pages.

Martucci, Stephen A., Reversible Compression of HDTV Images Using Median Adaptive Prediction and Arithmetic Coding, IEEE, 1990, 4 pages.

International Search Report, App. No. PCT/US2013/063233 dated Jan. 10, 2014, 6 pages.

Partial European Search Report, App. No. 13004799.6 dated Apr. 7, 2014, 5 pages.

* cited by examiner

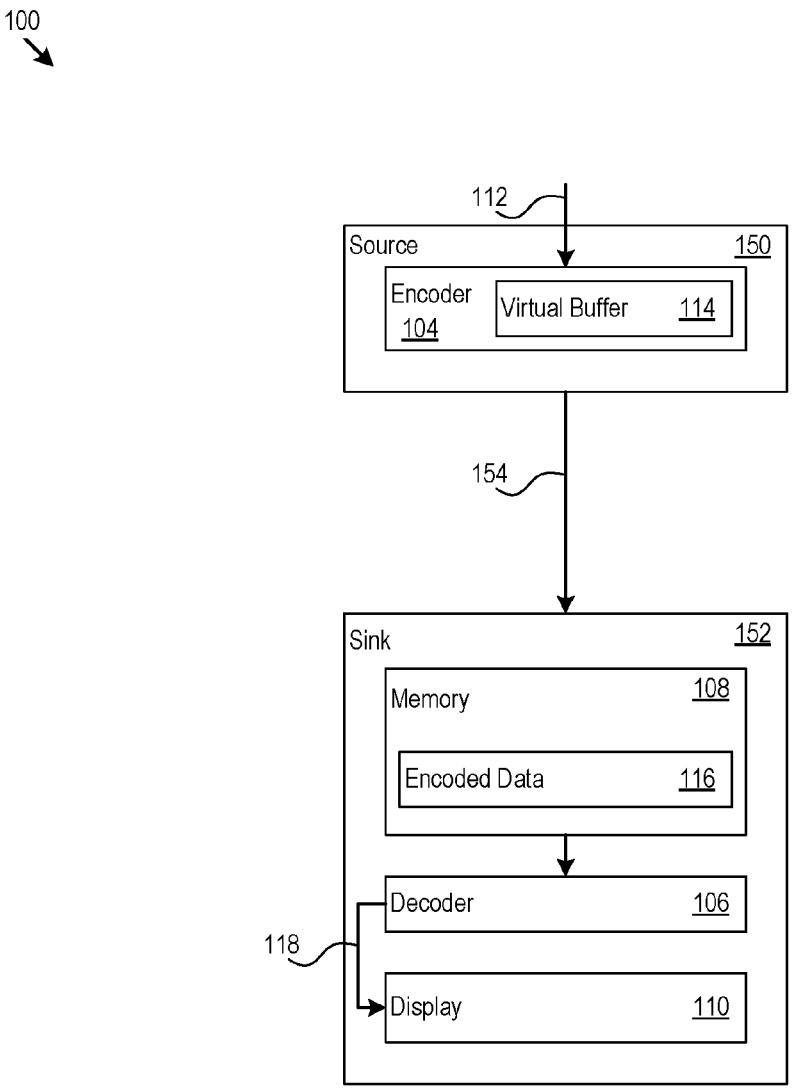

100
112
Source
150
Encoder
104
Virtual Buffer
114
154
Sink
152
Memory
108
Encoded Data
116
Decoder
106
118
Display
110

Figure 1

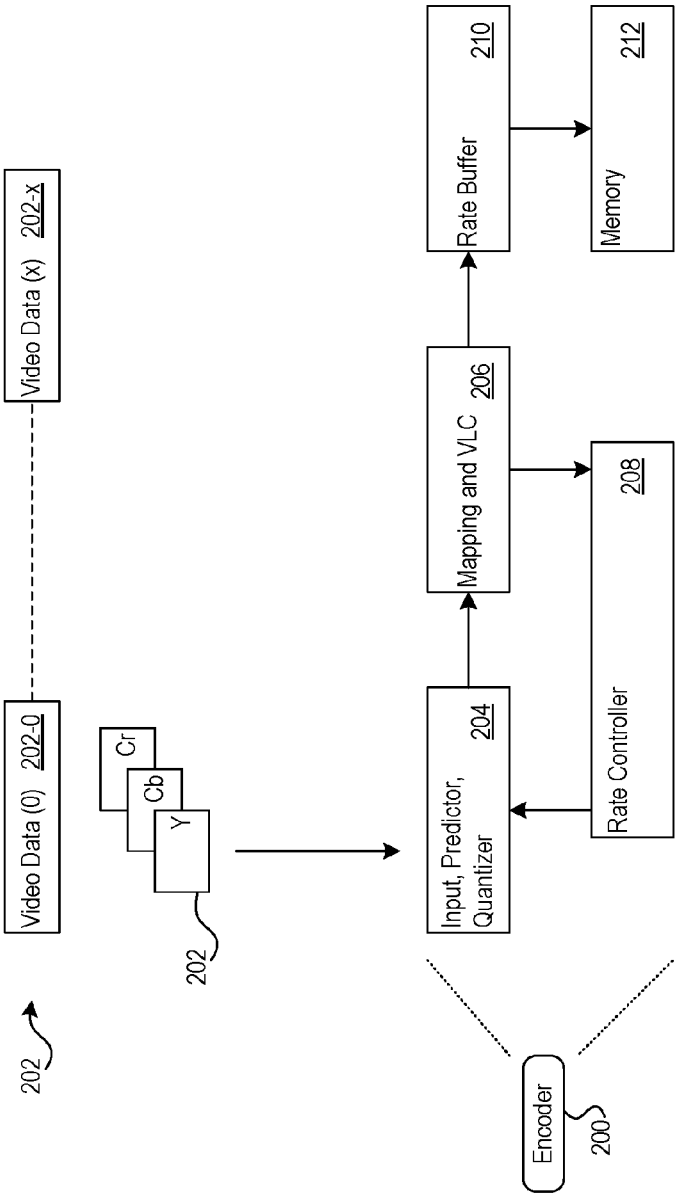

Video Data (0) 202-0
Video Data (x) 202-x
202
Cr
Cb
Y
202
Input, Predictor, Quantizer 204
Mapping and VLC 206
Rate Buffer 210
Memory 212
Rate Controller 208
Encoder
200

Figure 2

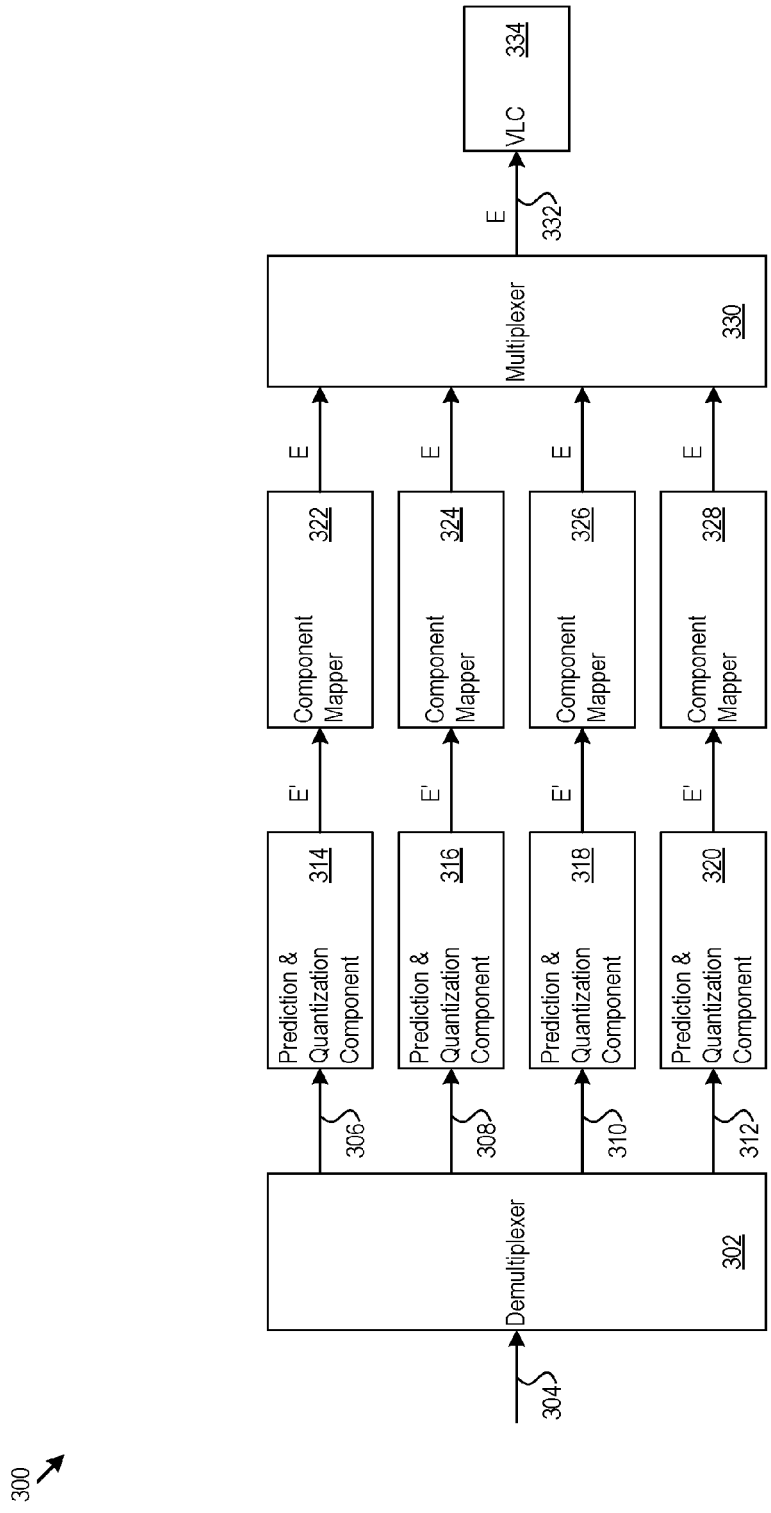

300
304
Demultiplexer
302
306
308
310
312
Prediction & Quantization Component
314
316
318
320
E'
Component Mapper
322
324
326
328
E
Multiplexer
330
332
VLC
334

Figure 3

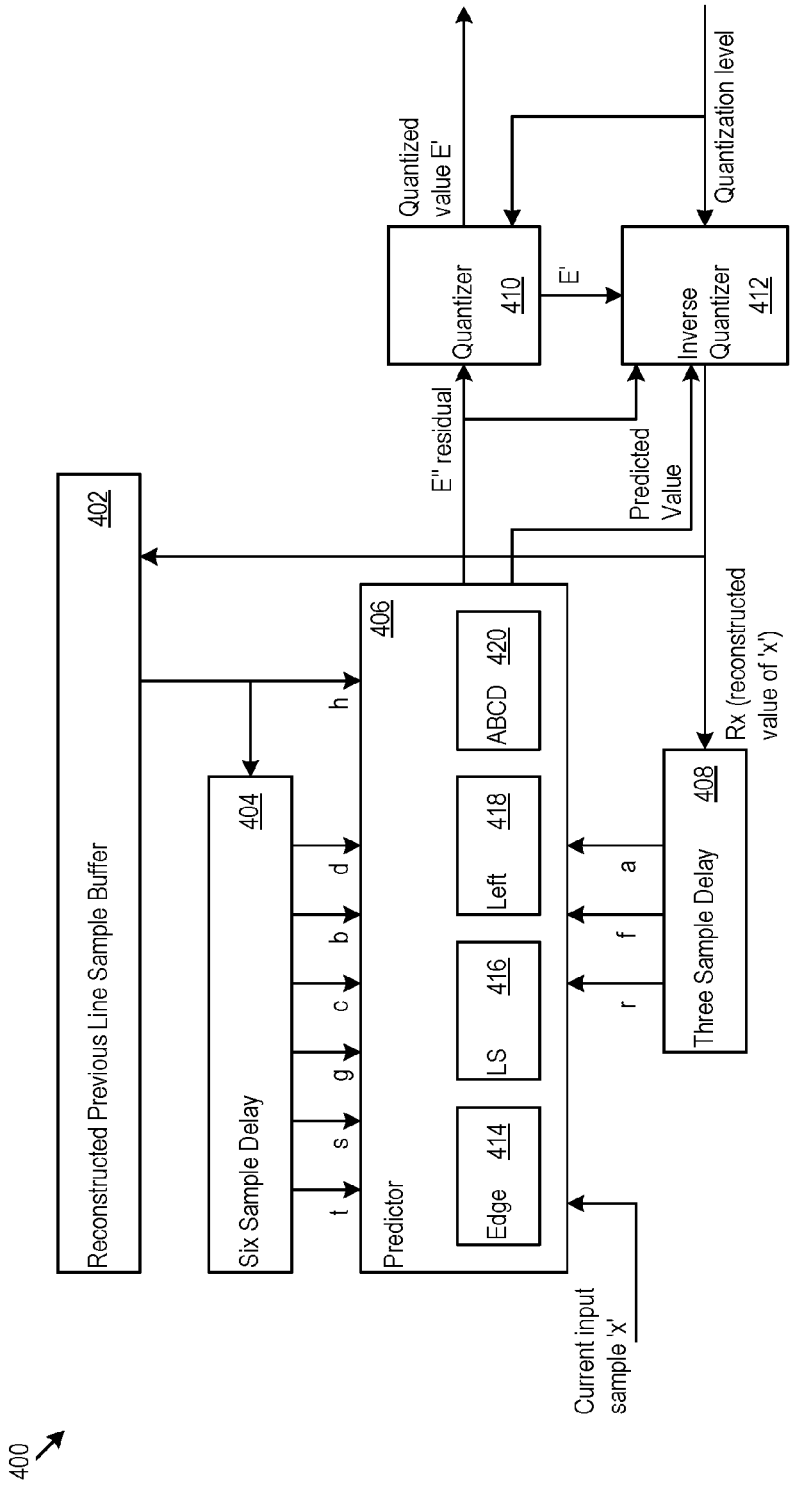

400
Reconstructed Previous Line Sample Buffer
402
Six Sample Delay
404
Predictor
406
Edge 414
LS 416
Left 418
ABCD 420
Three Sample Delay
408
Quantizer 410
Inverse Quantizer 412
Quantized value E'
Quantization level
E'
E'' residual
Predicted Value
Rx (reconstructed value of 'x')
Current input sample 'x'
h
d
b
c
g
s
t
r
f
a

Figure 4

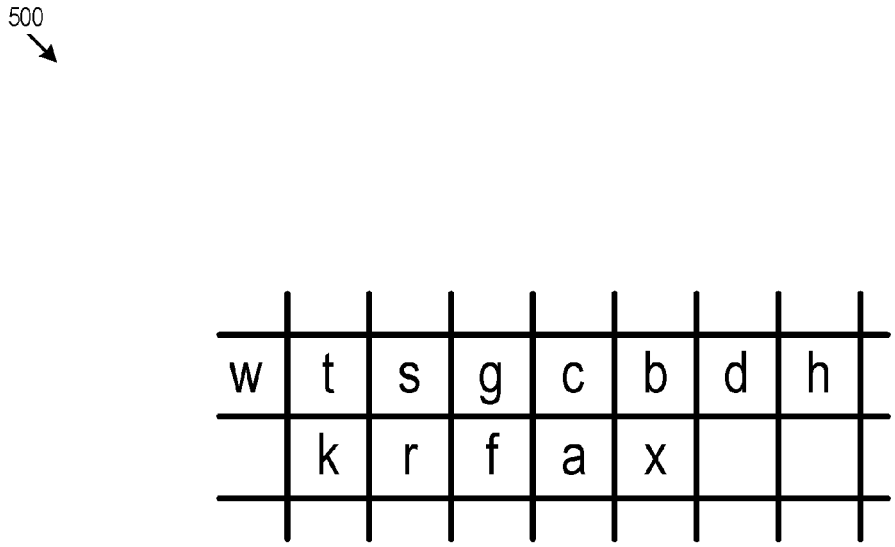


Figure 5

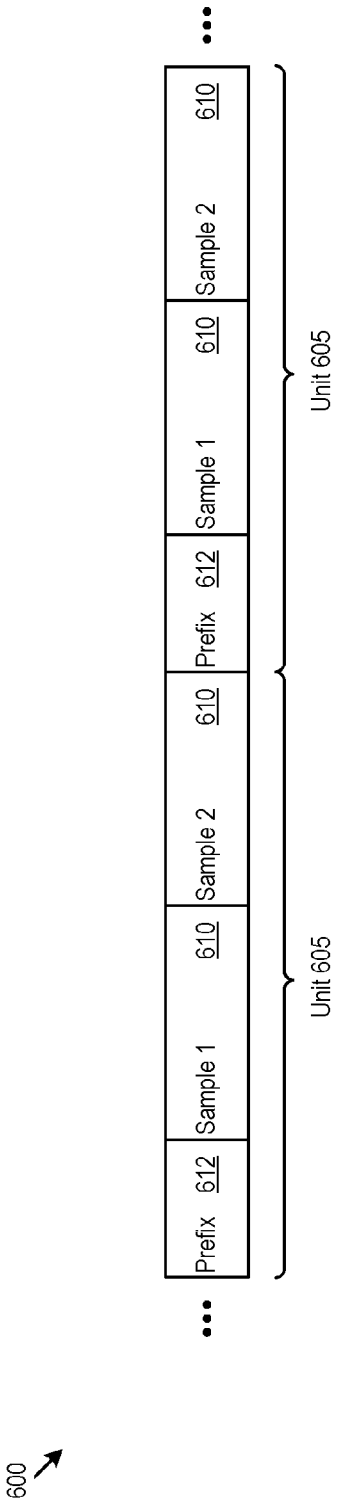

600
Prefix 612
Sample 1
610
Sample 2
610
Prefix 612
Sample 1
610
Sample 2
610
Unit 605
Unit 605

Figure 6

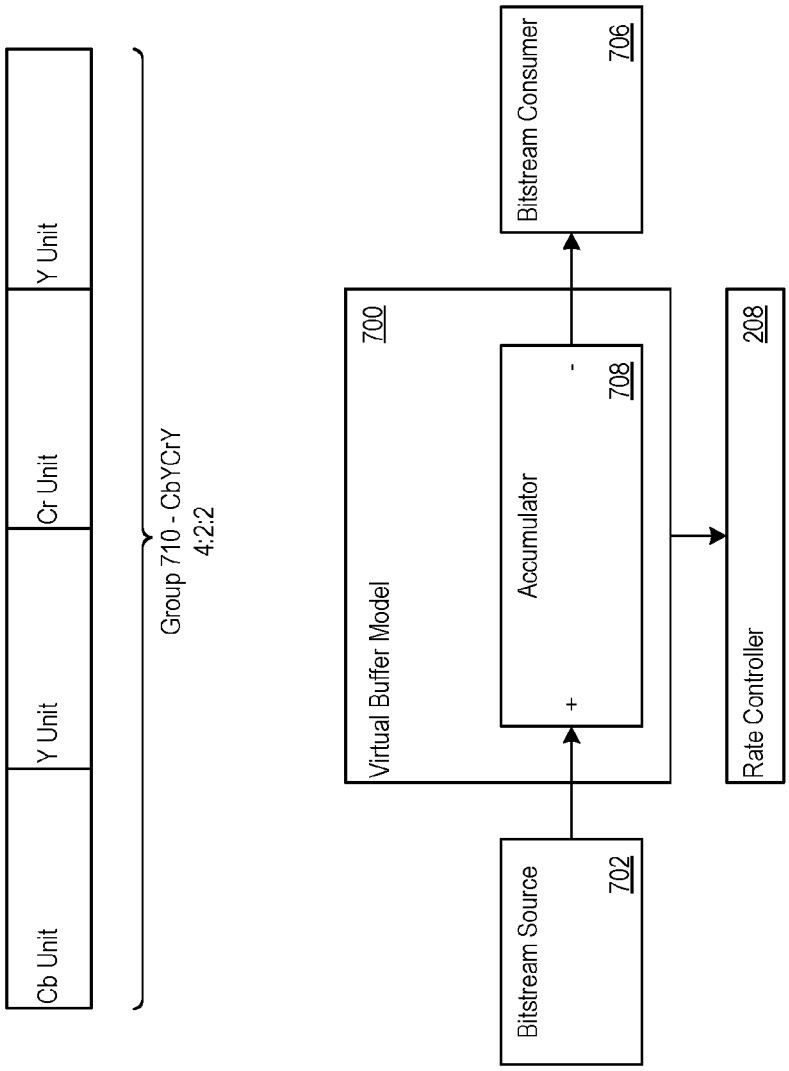

Cb Unit
Y Unit
Cr Unit
Y Unit
Group 710 - CbYCrY
4:2:2
Bitstream Source
702
Virtual Buffer Model
700
Accumulator
708
+
-
Bitstream Consumer
706
Rate Controller
208

Figure 7

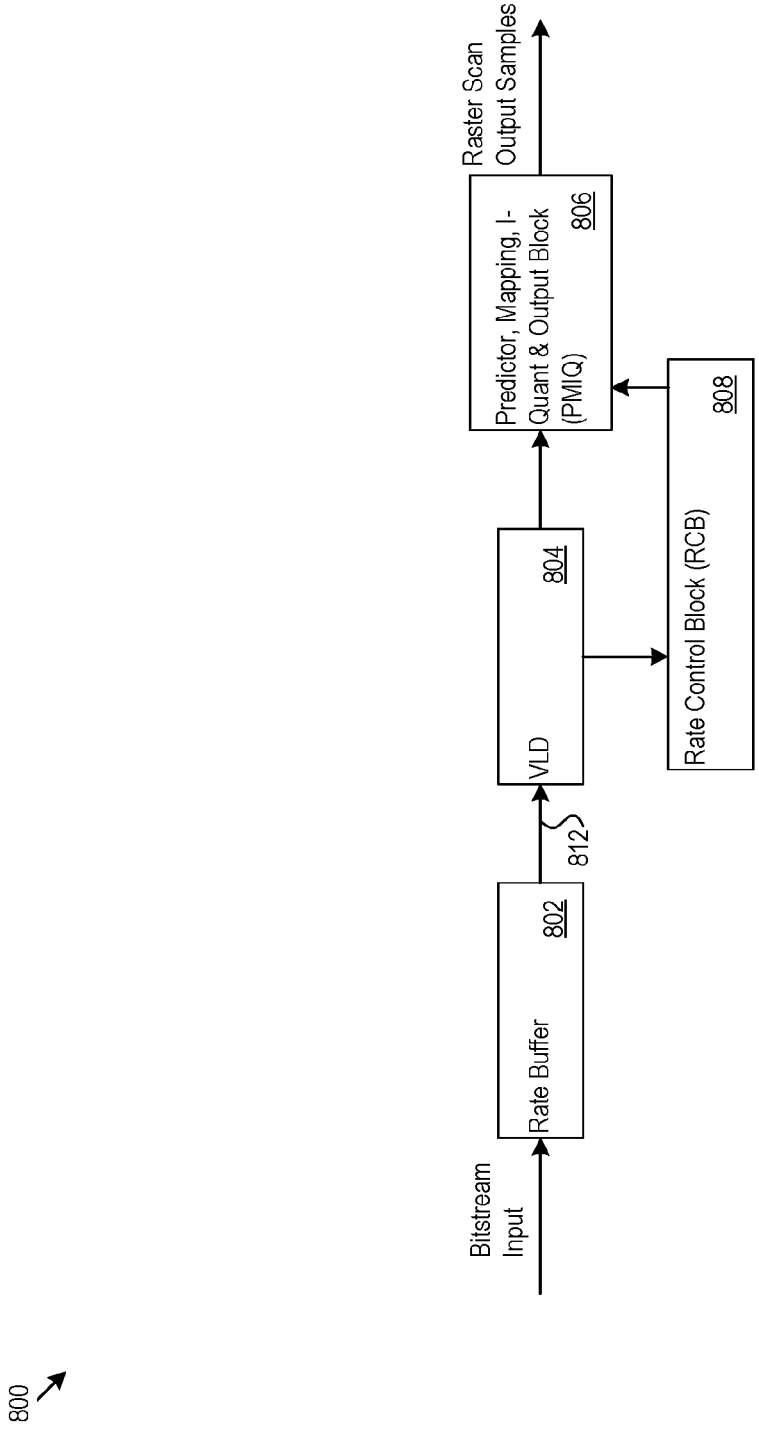

800
Bitstream Input
Rate Buffer
802
812
VLD
804
Rate Control Block (RCB)
808
Predictor, Mapping, I-Quant & Output Block (PMIQ)
806
Raster Scan Output Samples

Figure 8

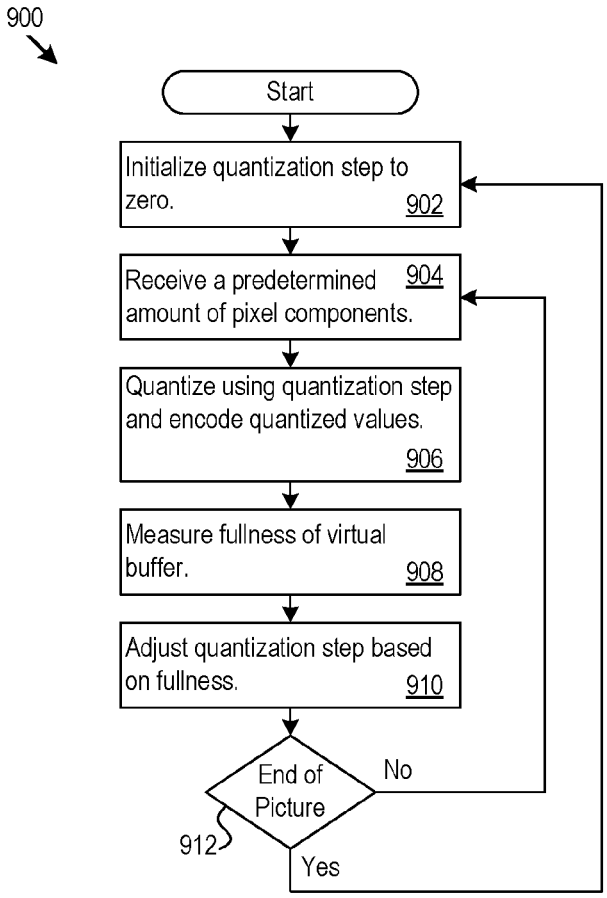

900
Start
Initialize quantization step to zero. 902
Receive a predetermined amount of pixel components. 904
Quantize using quantization step and encode quantized values. 906
Measure fullness of virtual buffer. 908
Adjust quantization step based on fullness. 910
End of Picture
912
No
Yes

Figure 9

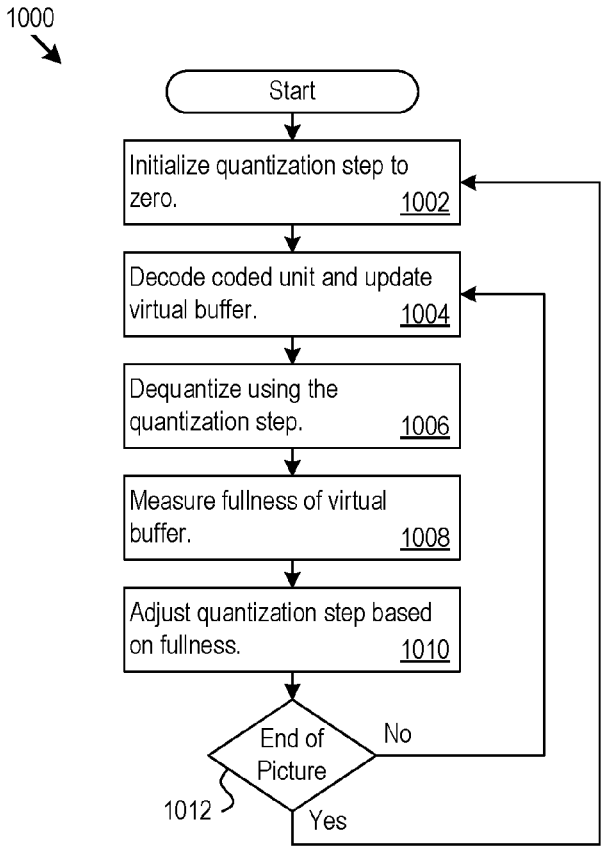

1000
Start
Initialize quantization step to zero. 1002
Decode coded unit and update virtual buffer. 1004
Dequantize using the quantization step. 1006
Measure fullness of virtual buffer. 1008
Adjust quantization step based on fullness. 1010
End of Picture
No
Yes
1012

Figure 10

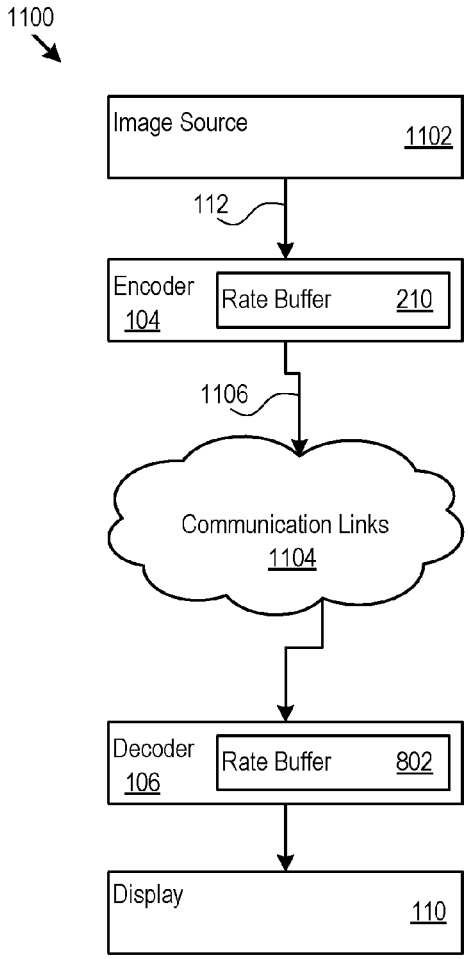

1100
Image Source
1102
112
Encoder
104
Rate Buffer
210
1106
Communication Links
1104
Decoder
106
Rate Buffer
802
Display
110

Figure 11

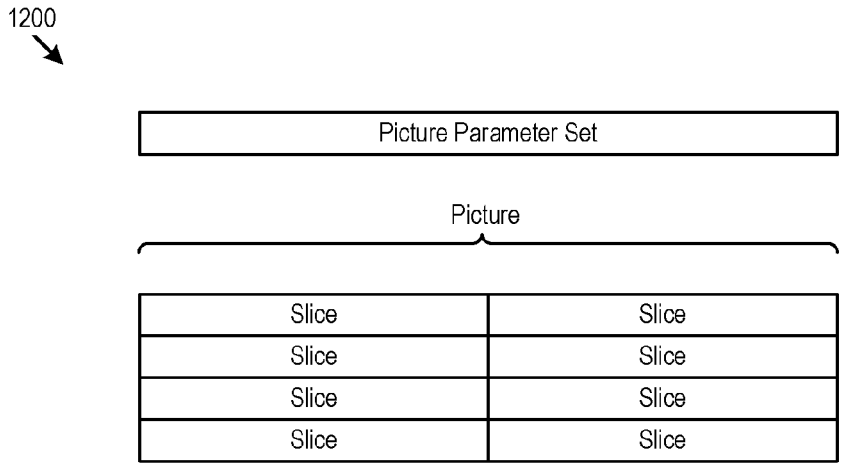

1200
Picture Parameter Set
Picture
Slice
Slice
Slice
Slice
Slice
Slice
Slice
Slice

Figure 12

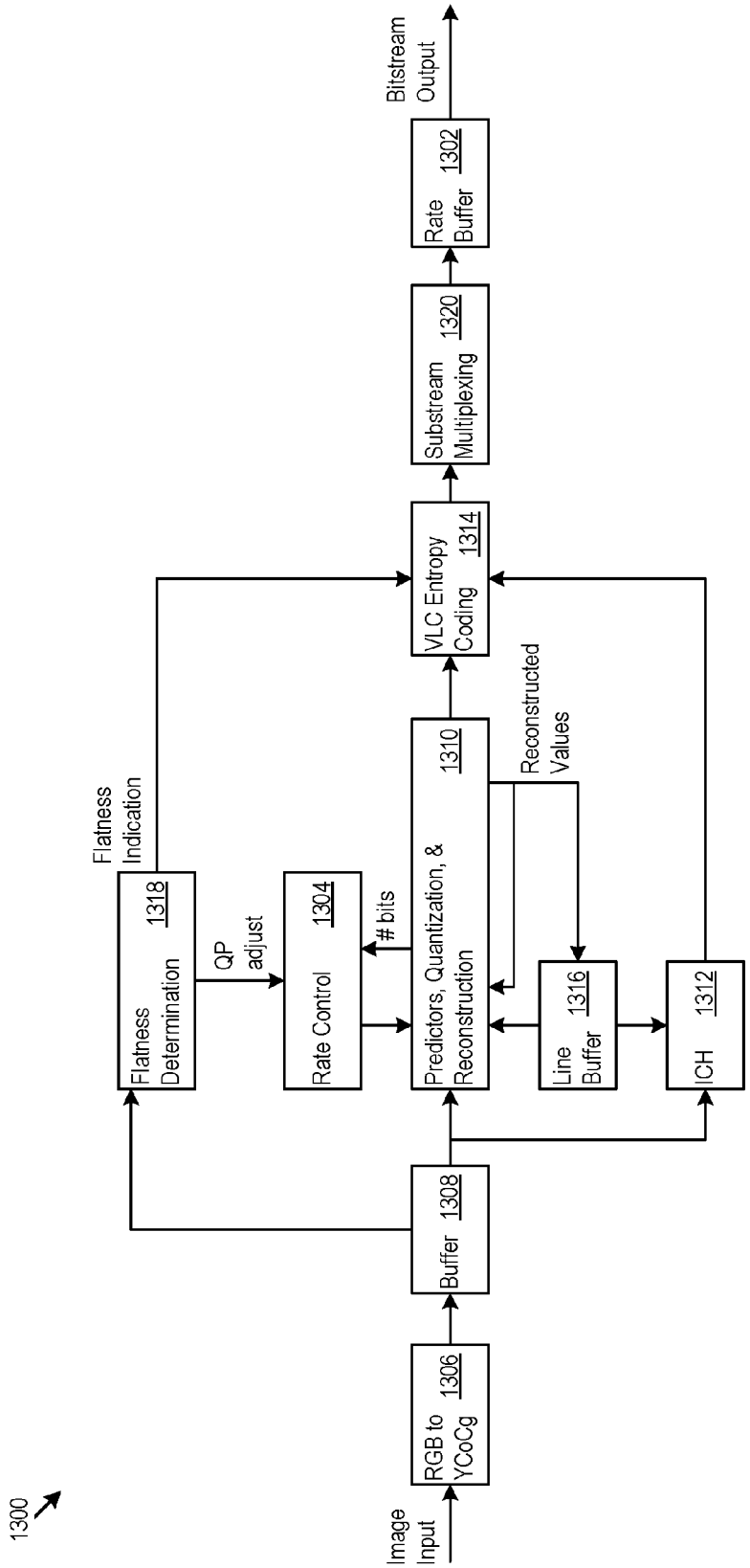

1300
Image Input
RGB to YCoCg 1306
Buffer 1308
Flatness Determination 1318
QP adjust
Rate Control 1304
bits
Predictors, Quantization, & Reconstruction 1310
Reconstructed Values
Line Buffer 1316
ICH 1312
Flatness Indication
VLC Entropy Coding 1314
Substream Multiplexing 1320
Rate Buffer 1302
Bitstream Output

Figure 13

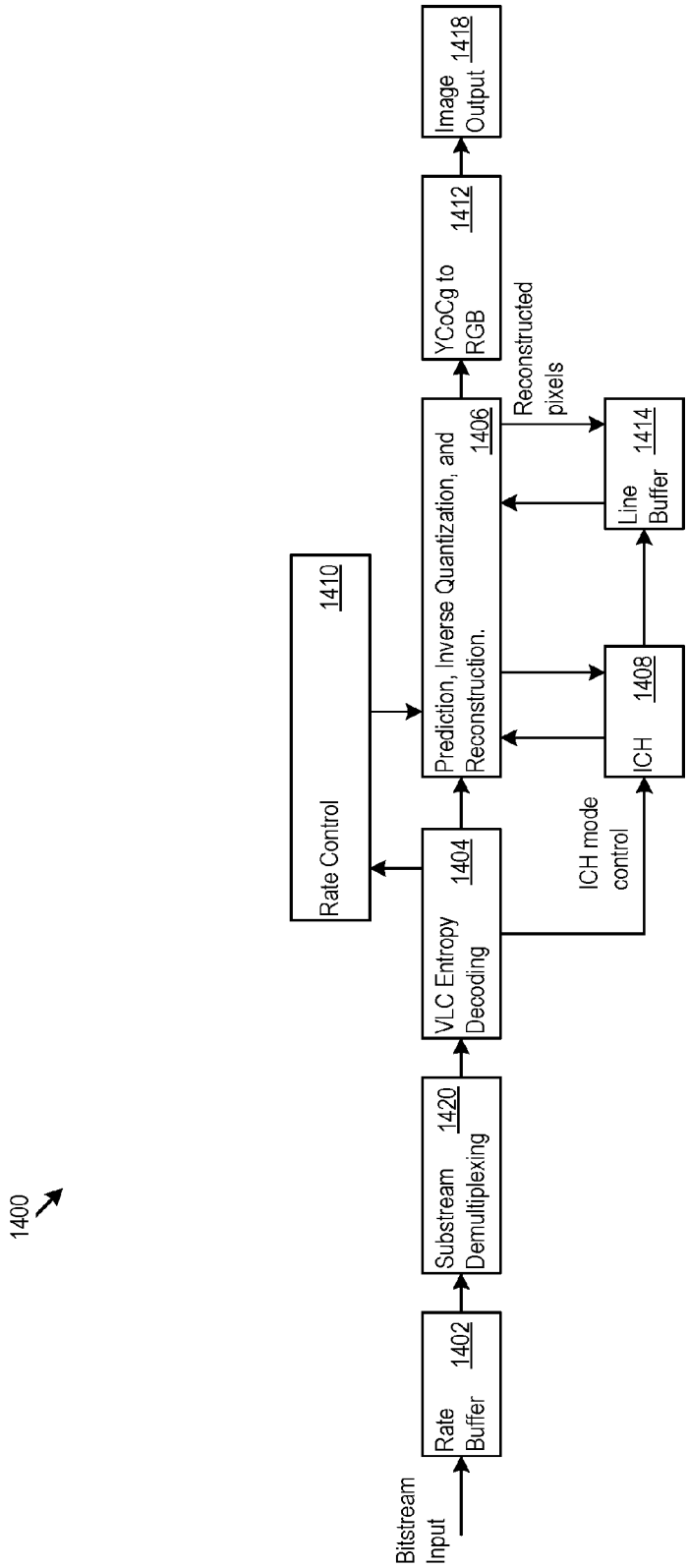

1400
Bitstream Input
Rate Buffer 1402
Substream Demultiplexing 1420
VLC Entropy Decoding 1404
Rate Control 1410
ICH mode control
Prediction, Inverse Quantization, and Reconstruction. 1406
ICH 1408
Line Buffer 1414
Reconstructed pixels
YCoCg to RGB 1412
Image Output 1418

Figure 14

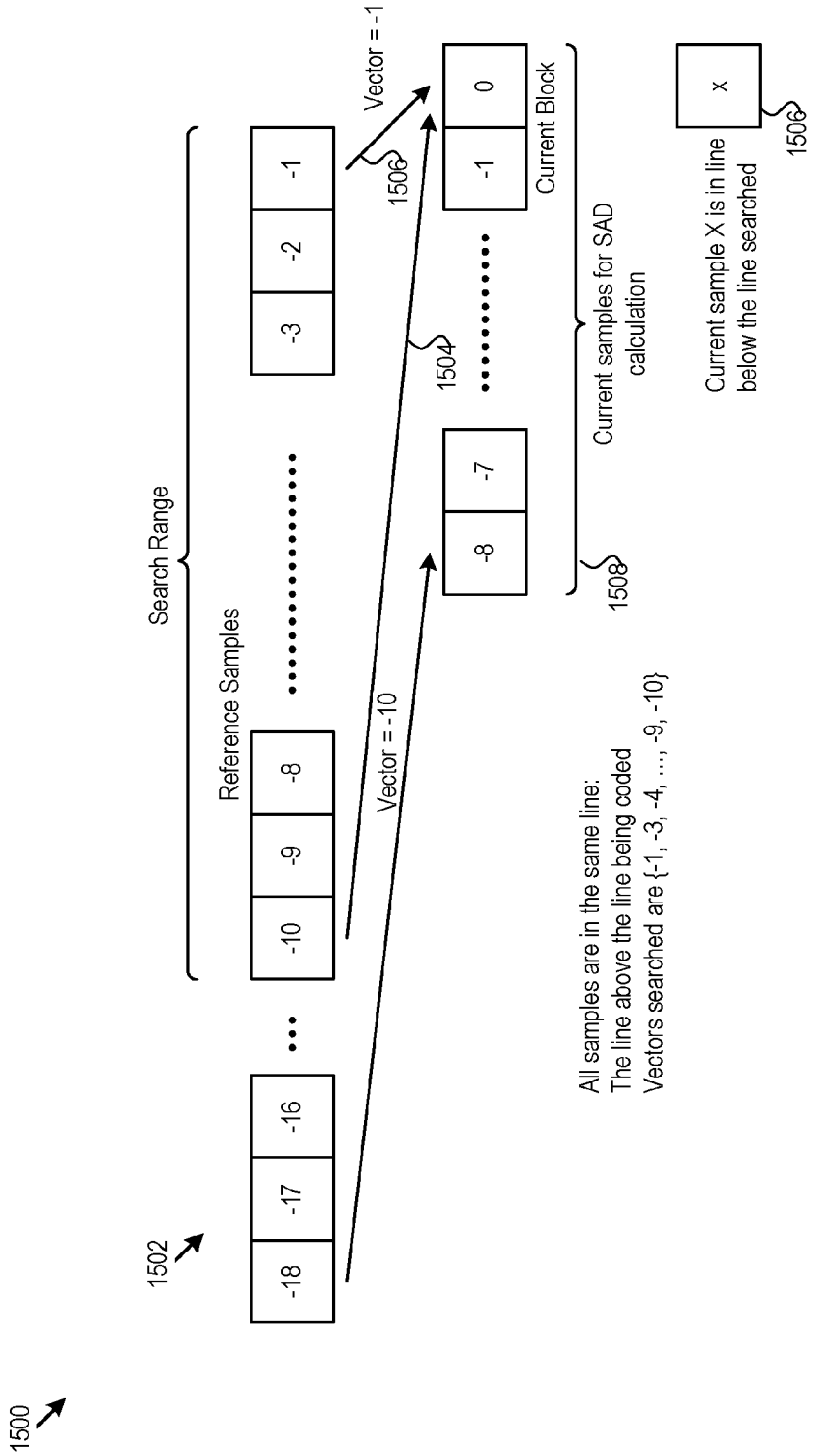

1500
1502
-18
-17
-16
...
-10
-9
-8
Reference Samples
Search Range
Vector = -10
-8
-7
.....
-3
-2
-1
1504
Vector = -1
1506
-1
0
Current Block
1508
Current samples for SAD calculation
All samples are in the same line:
The line above the line being coded
Vectors searched are {-1, -3, -4, ..., -9, -10}
x
Current sample X is in line below the line searched
1506

Figure 15

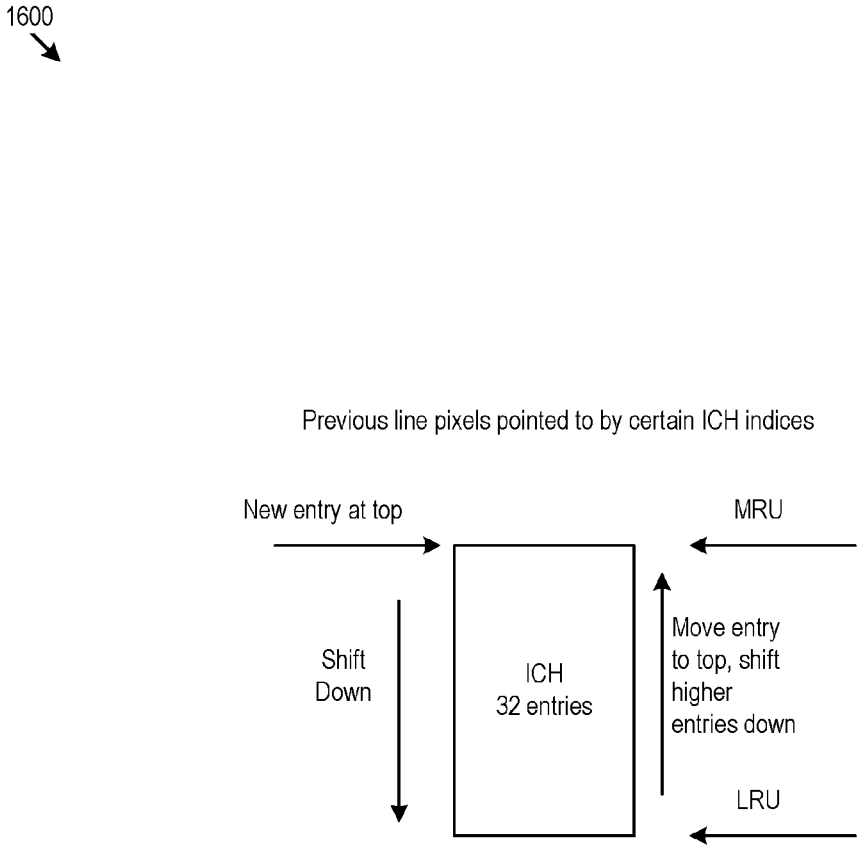

1600
Previous line pixels pointed to by certain ICH indices
New entry at top
MRU
Shift
Down
ICH
32 entries
Move entry
to top, shift
higher
entries down
LRU

Figure 16

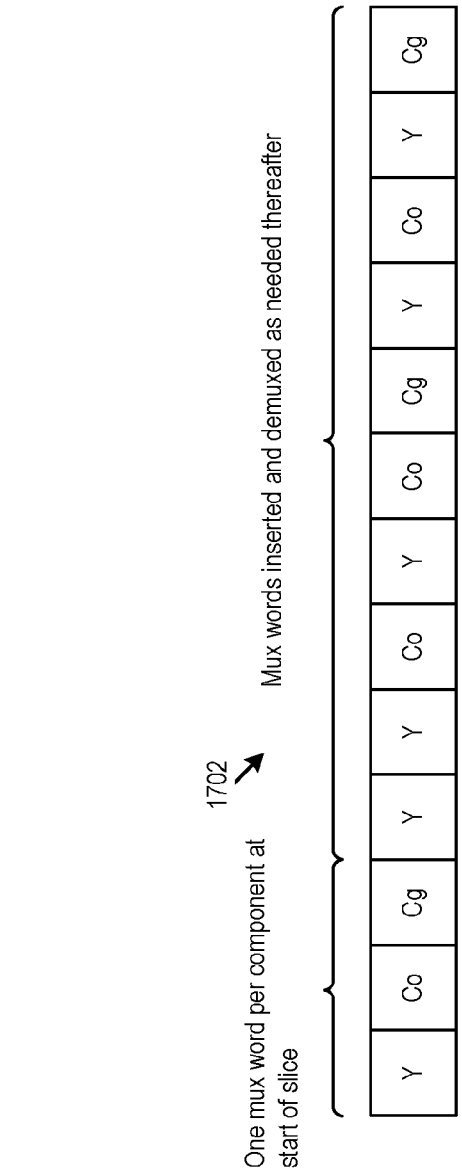

1700
1702
One mux word per component at start of slice
Mux words inserted and demuxed as needed thereafter
Y
Co
Cg
Y
Y
Co
Y
Co
Cg
Y
Co
Y
Cg
Mux words @ 48 bits
In this example, Y uses more bits than Co, which uses more bits than Cg

Figure 17

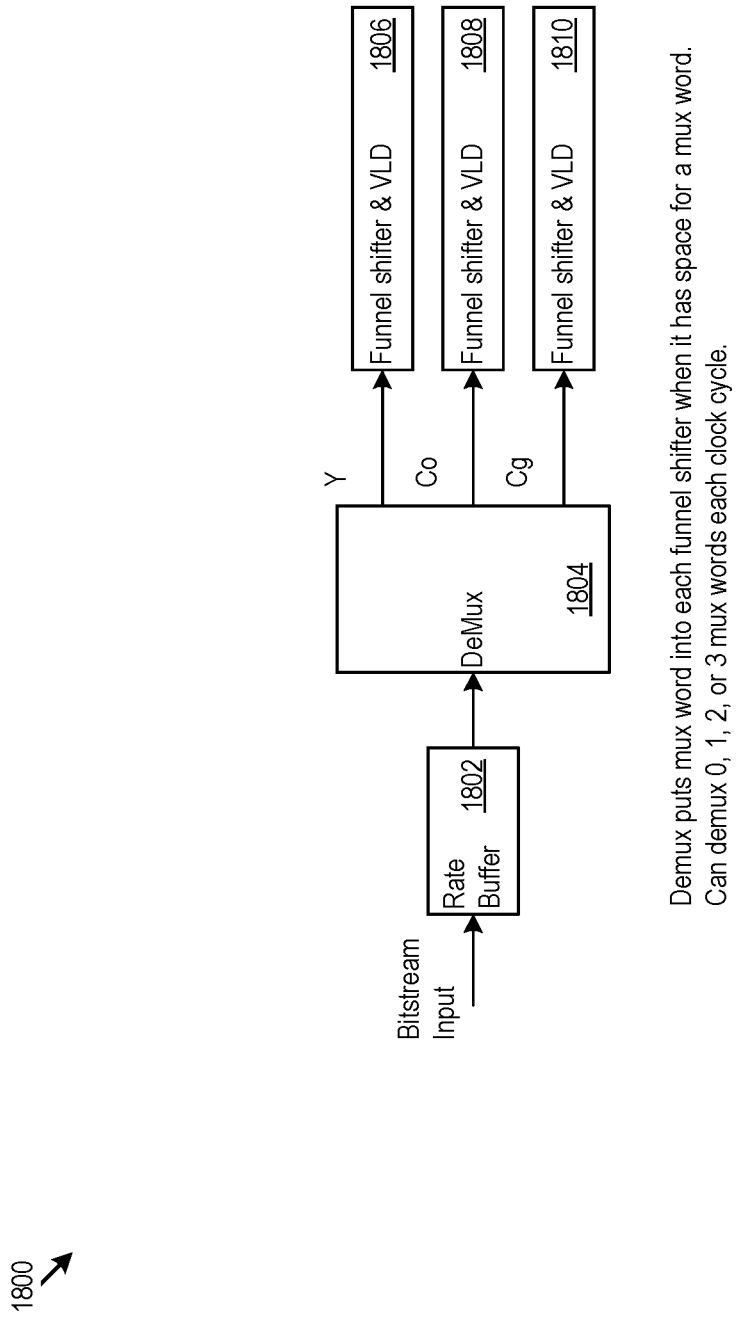
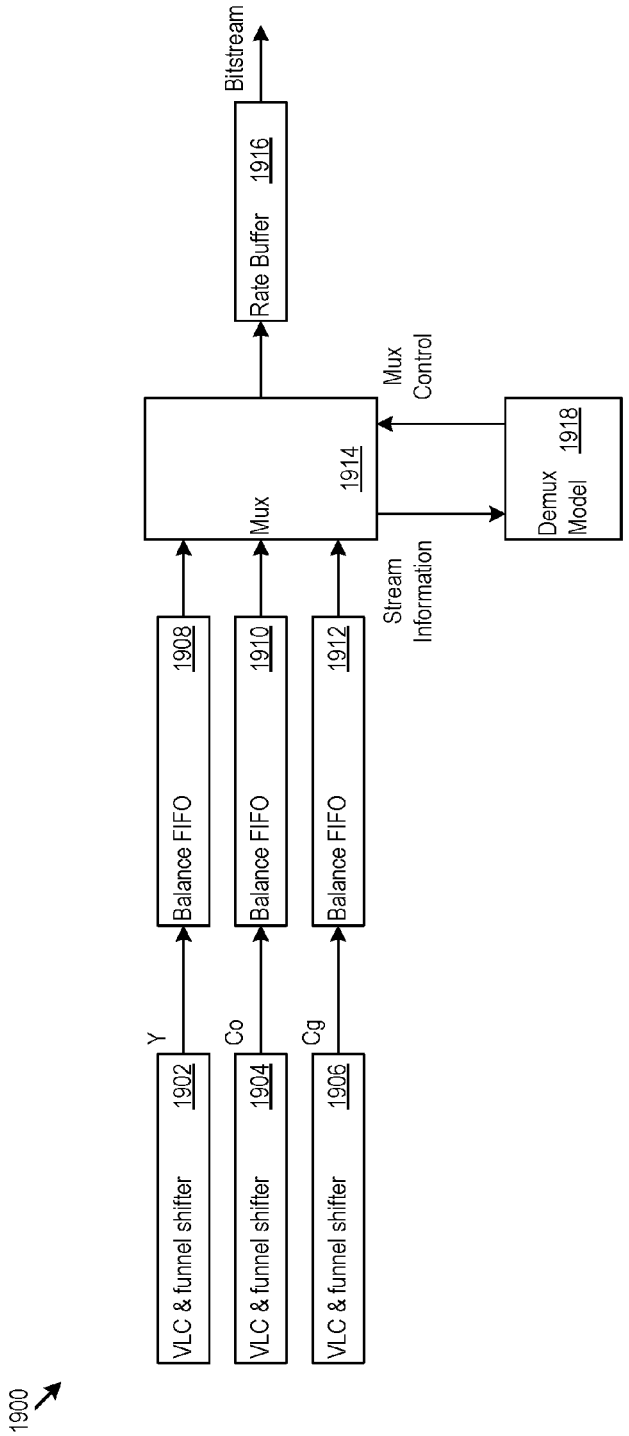

1800
Bitstream Input
Rate Buffer 1802
DeMux 1804
Y
Co
Cg
Funnel shifter & VLD 1806
Funnel shifter & VLD 1808
Funnel shifter & VLD 1810
Demux puts mux word into each funnel shifter when it has space for a mux word.
Can demux 0, 1, 2, or 3 mux words each clock cycle.

Figure 18



Mux puts mux word from each substream into bitstream when its model of decoder funnel shifter has space for a mux word. Can mux 0, 1, 2, or 3 mux words each clock cycle.

Figure 19

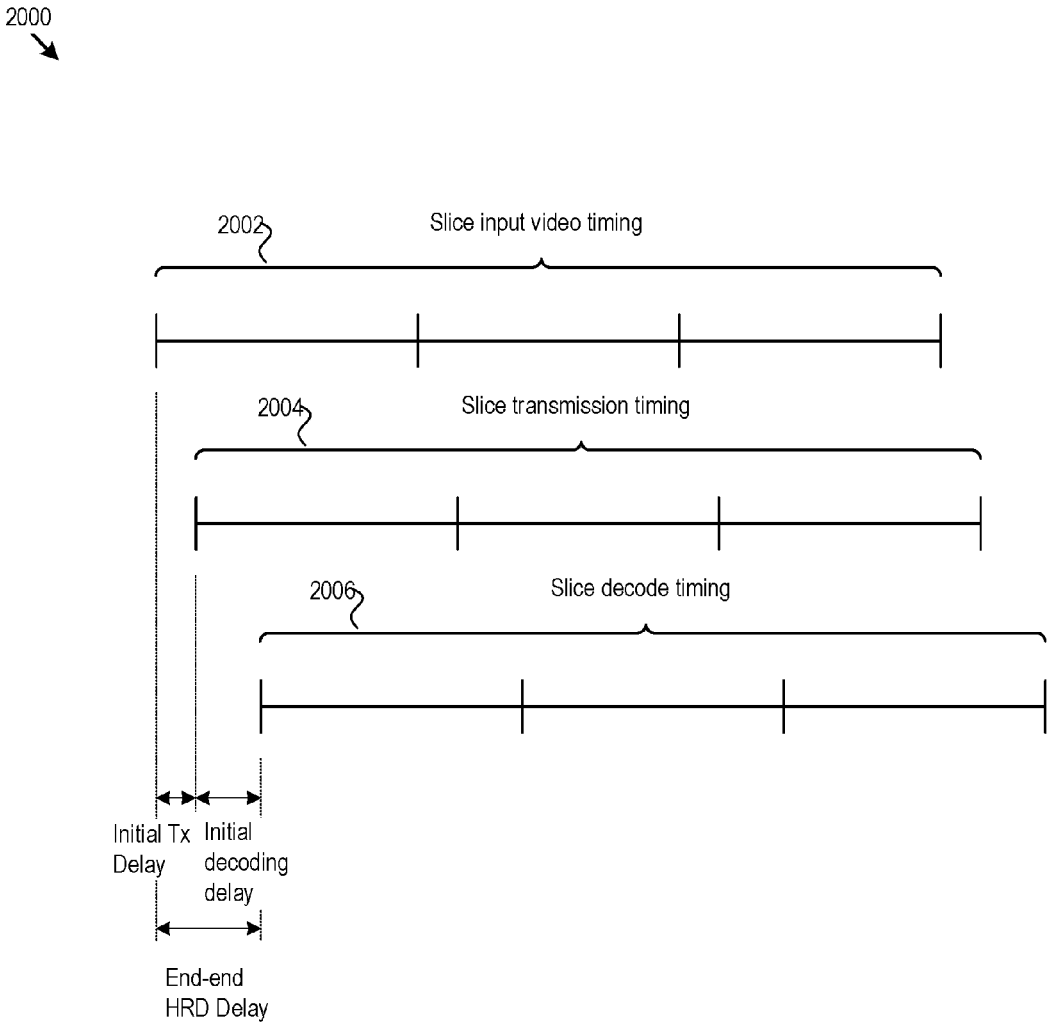

2000
2002
Slice input video timing
2004
Slice transmission timing
2006
Slice decode timing
Initial Tx Delay
Initial decoding delay
End-end HRD Delay

Figure 20

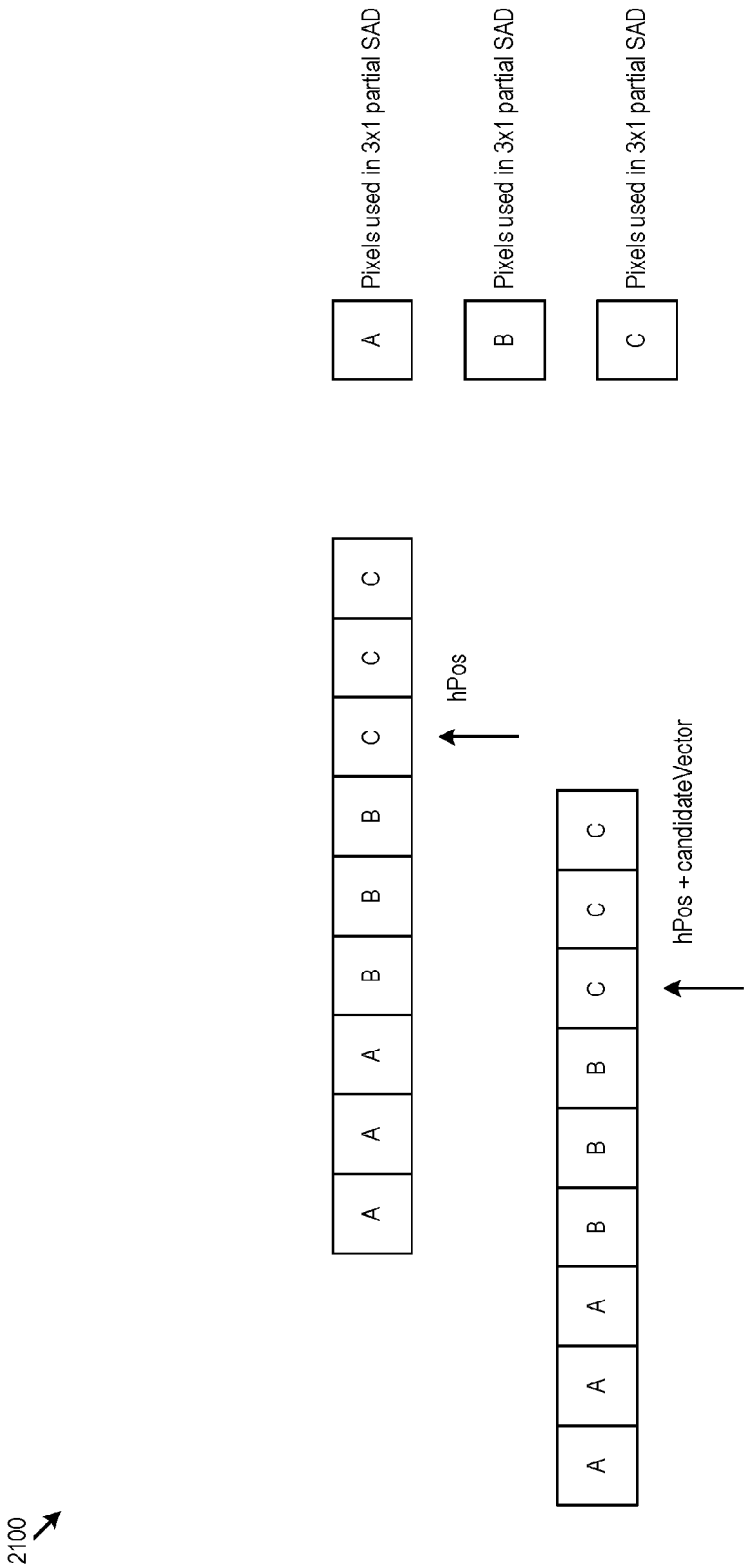

Pixels used in 3x1 partial SAD
A
Pixels used in 3x1 partial SAD
B
Pixels used in 3x1 partial SAD
C
A A A B B B C C C
hPos
A A A B B B C C C
hPos + candidateVector
2100

Figure 21

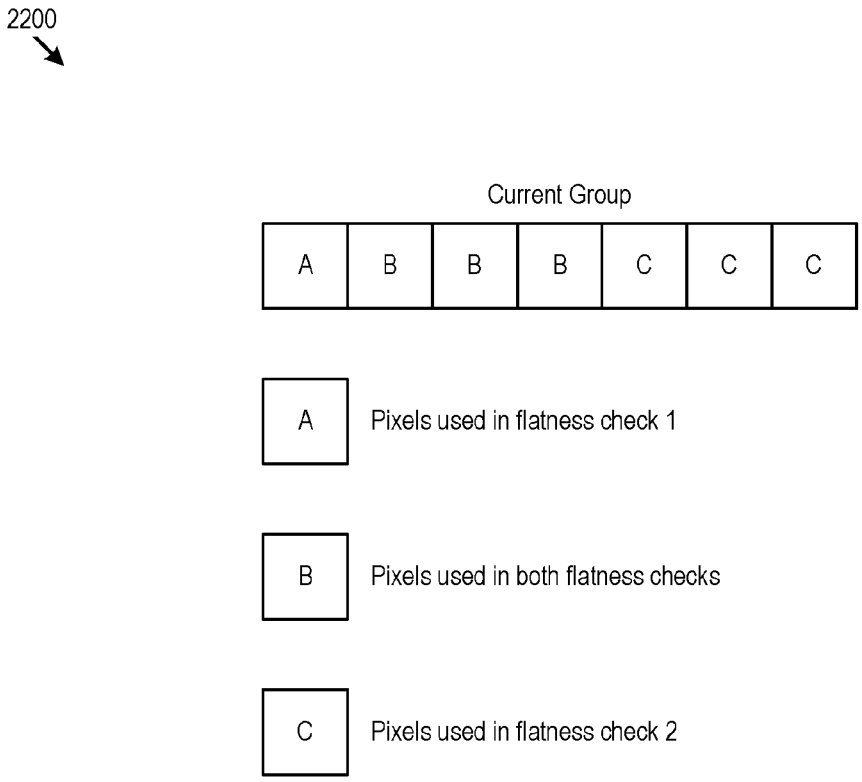

2200
Current Group
A
B
B
B
C
C
C
A
Pixels used in flatness check 1
B
Pixels used in both flatness checks
C
Pixels used in flatness check 2

Figure 22

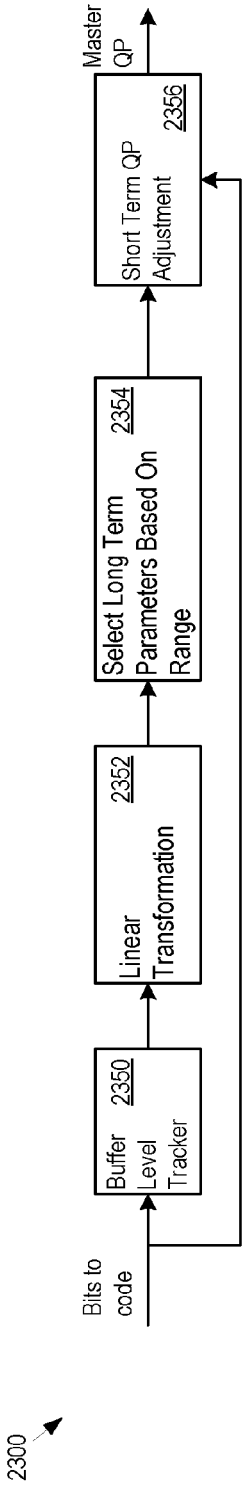

2300
Bits to code
Buffer Level Tracker 2350
Linear Transformation 2352
Select Long Term Parameters Based On Range 2354
Short Term QP Adjustment 2356
Master QP

Figure 23

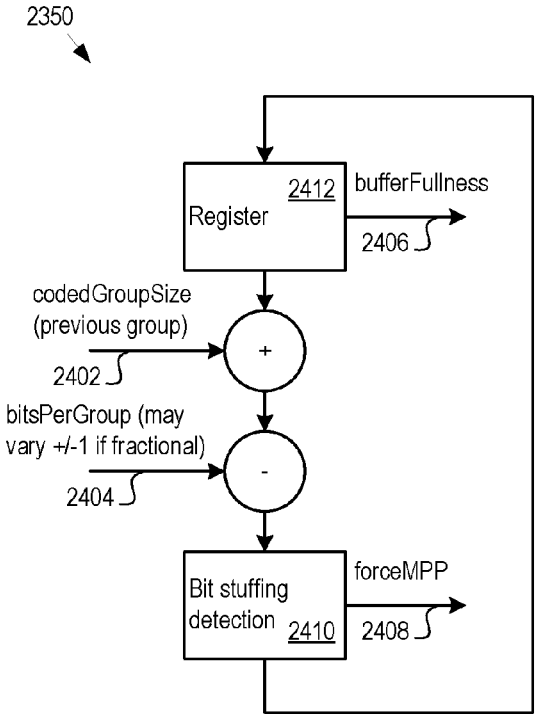

2350
Register
2412
bufferFullness
2406
codedGroupSize
(previous group)
2402
+
bitsPerGroup (may
vary +/-1 if fractional)
2404
-
Bit stuffing
detection
2410
forceMPP
2408

Figure 24

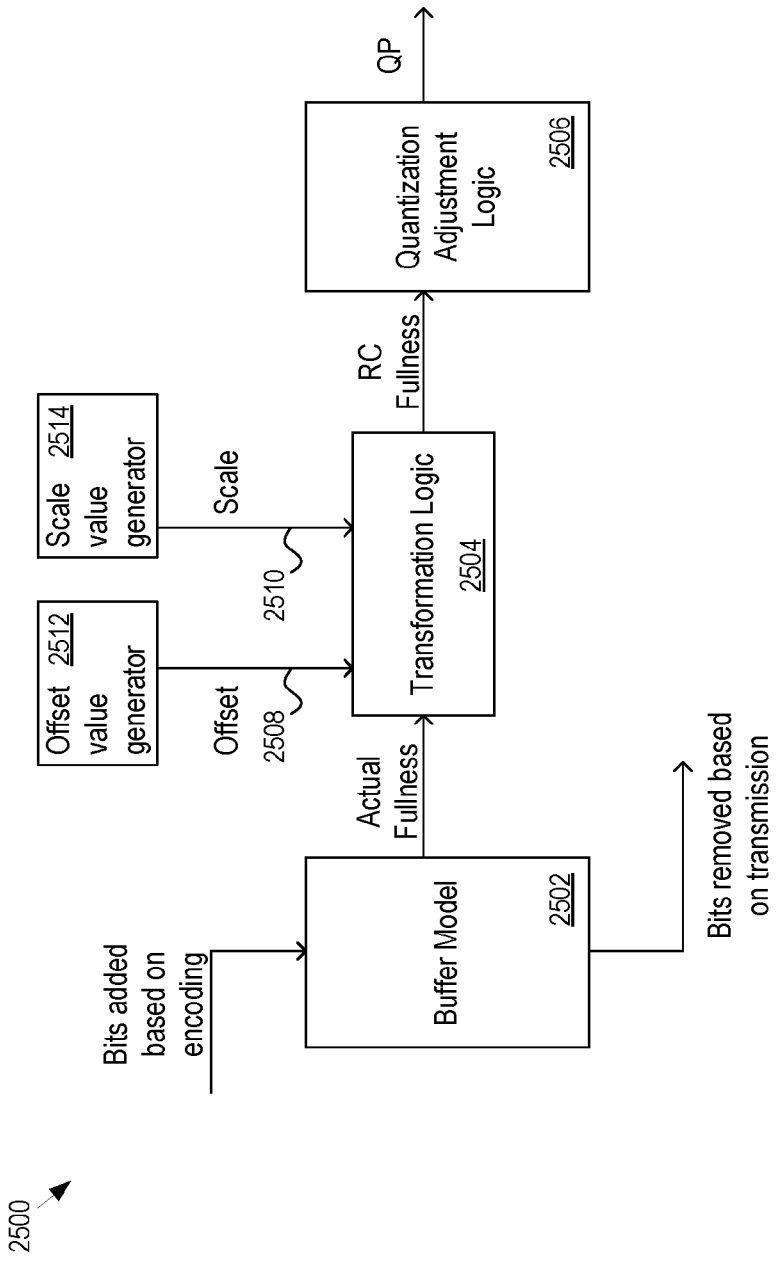

2500
Bits added based on encoding
Buffer Model
2502
Actual Fullness
Offset value generator 2512
Offset
2508
Scale value generator 2514
Scale
2510
Transformation Logic
2504
RC Fullness
Quantization Adjustment Logic
2506
QP
Bits removed based on transmission

Figure 25

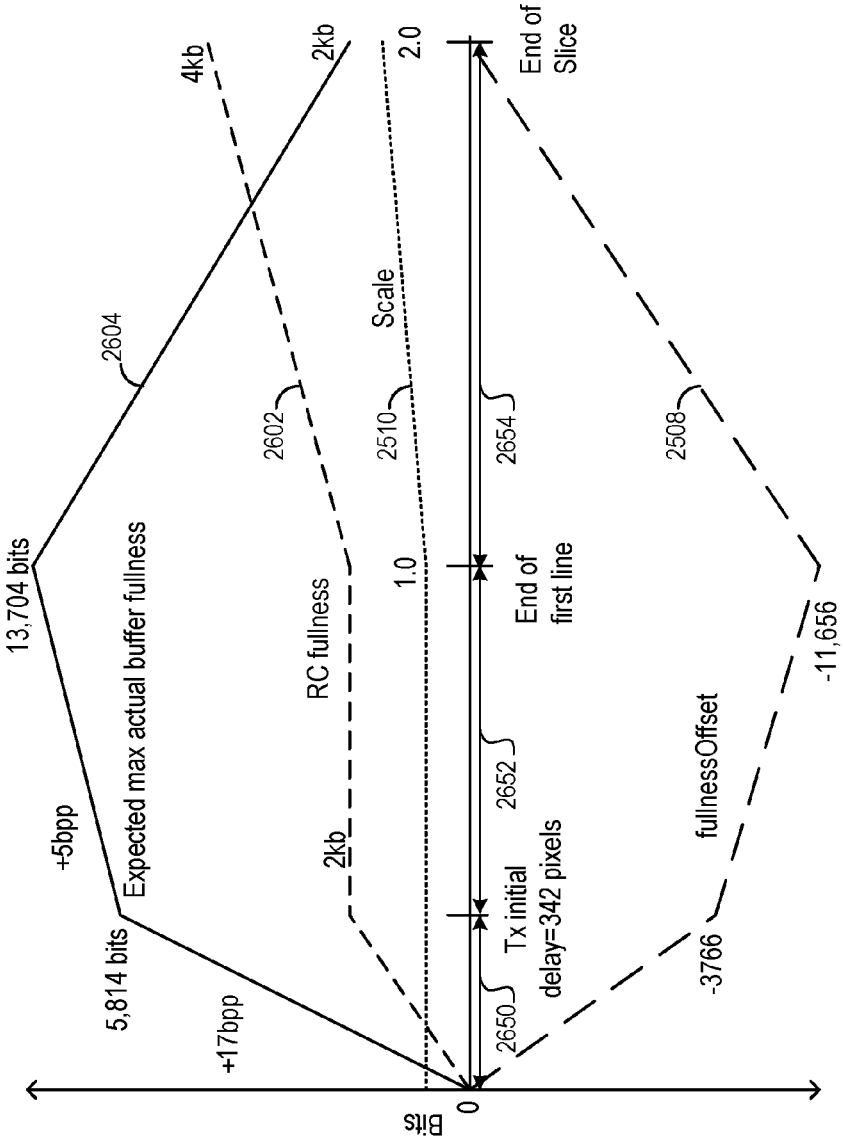


Figure 26

2600

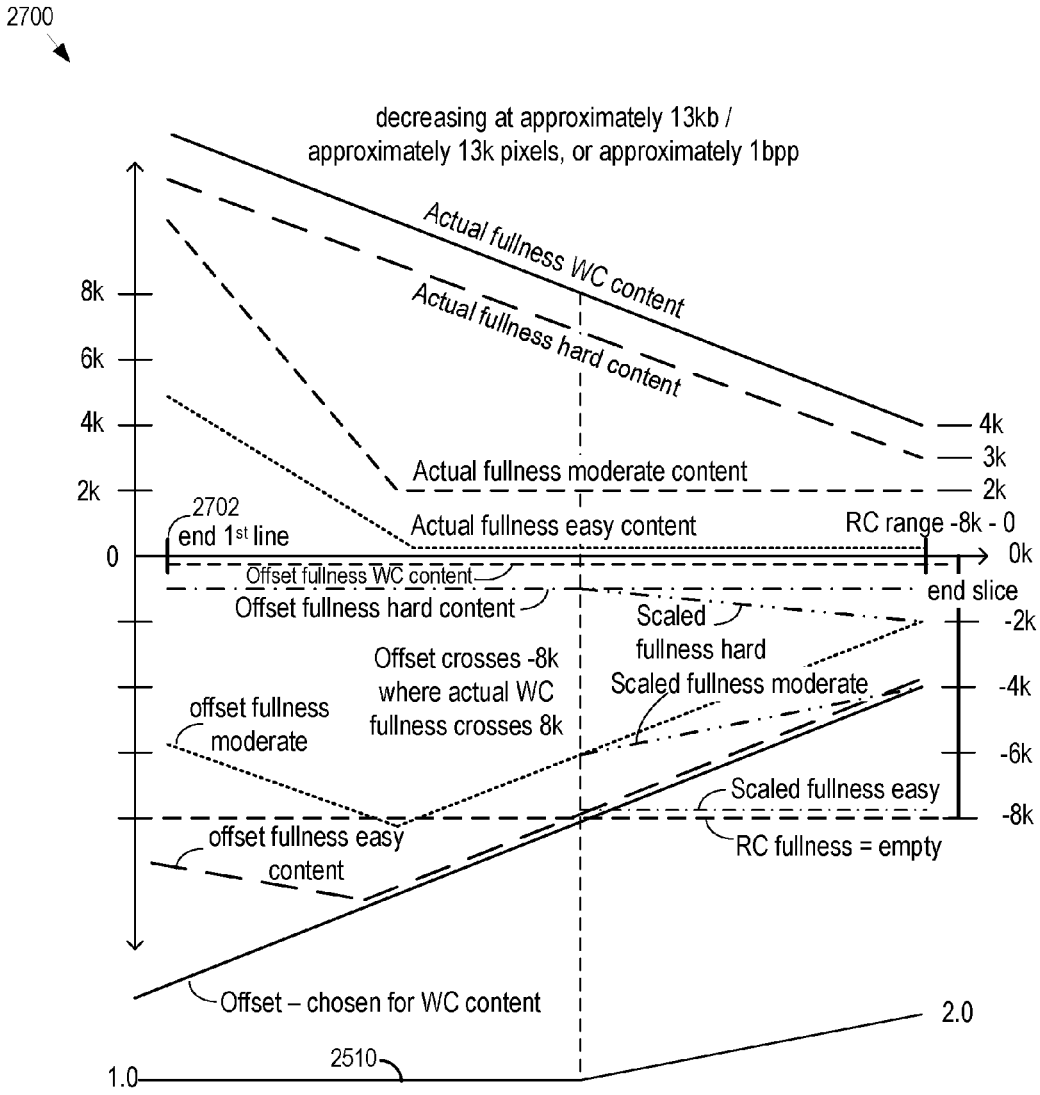

2700
decreasing at approximately 13kb /
approximately 13k pixels, or approximately 1bpp
Actual fullness WC content
Actual fullness hard content
Actual fullness moderate content
Actual fullness easy content
8k
6k
4k
2k
0
4k
3k
2k
0k
2702
end 1st line
RC range -8k - 0
Offset fullness WC content
Offset fullness hard content
end slice
Scaled
fullness hard
Scaled fullness moderate
-2k
-4k
-6k
-8k
Offset crosses -8k
where actual WC
fullness crosses 8k
offset fullness
moderate
Scaled fullness easy
RC fullness = empty
offset fullness easy
content
Offset – chosen for WC content
2.0
1.0
2510

Figure 27

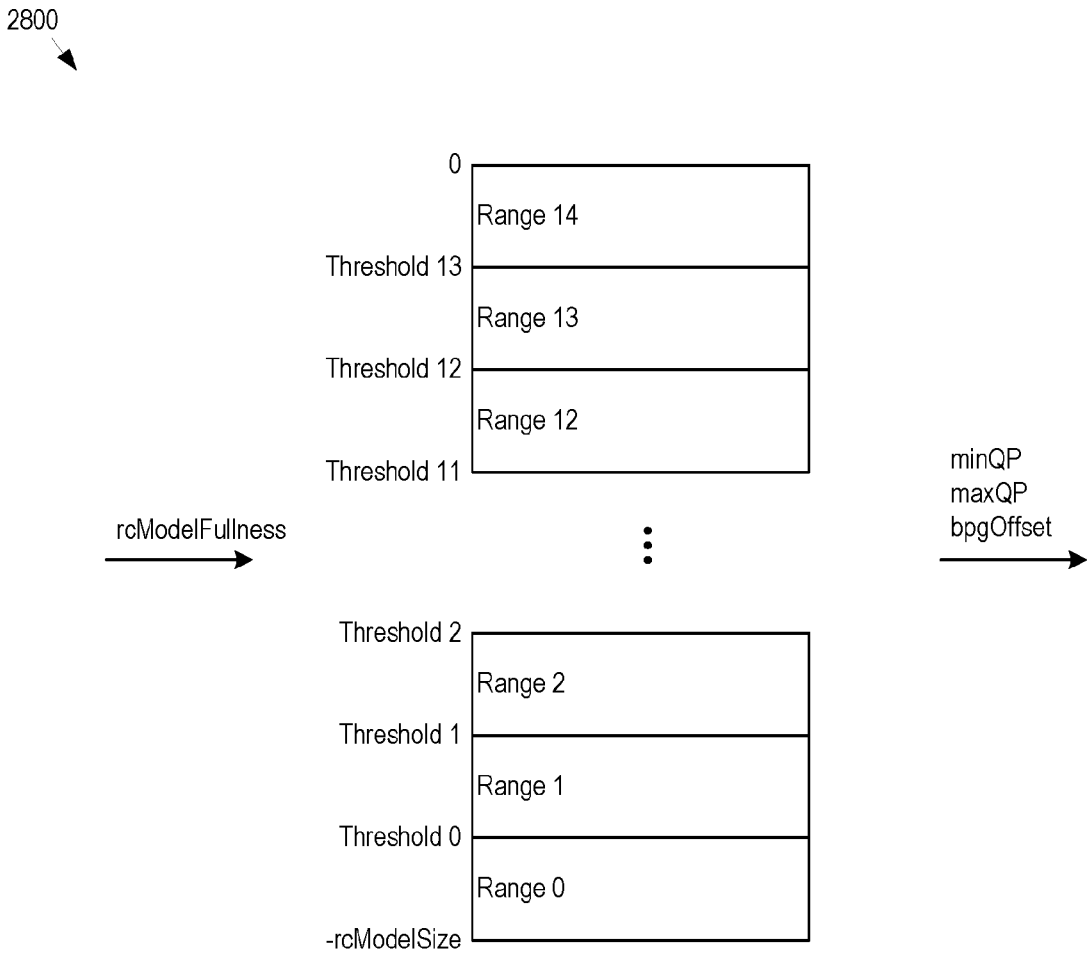

2800
0
Range 14
Threshold 13
Range 13
Threshold 12
Range 12
Threshold 11
rcModelFullness
minQP
maxQP
bpgOffset
Threshold 2
Range 2
Threshold 1
Range 1
Threshold 0
Range 0
-rcModelSize

Figure 28

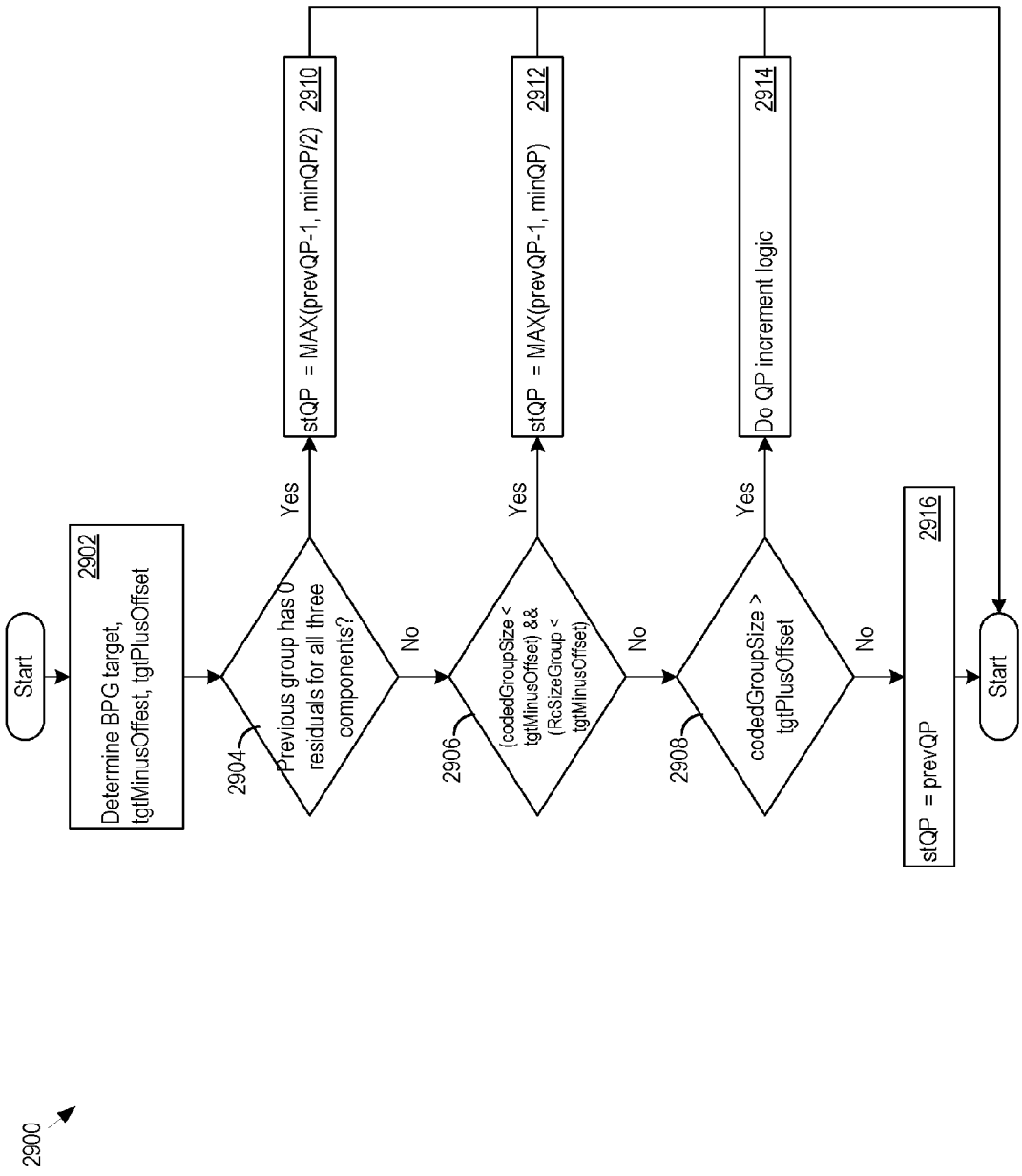

2900
Start
2902
Determine BPG target, tgtMinusOffest, tgtPlusOffset
2904
Previous group has 0 residuals for all three components?
Yes
stQP = MAX(prevQP-1, minQP/2) 2910
No
2906
(codedGroupSize < tgtMinusOffset) && (RcSizeGroup < tgtMinusOffset)
Yes
stQP = MAX(prevQP-1, minQP) 2912
No
2908
codedGroupSize > tgtPlusOffset
Yes
Do QP increment logic 2914
No
stQP = prevQP 2916
Start

Figure 29

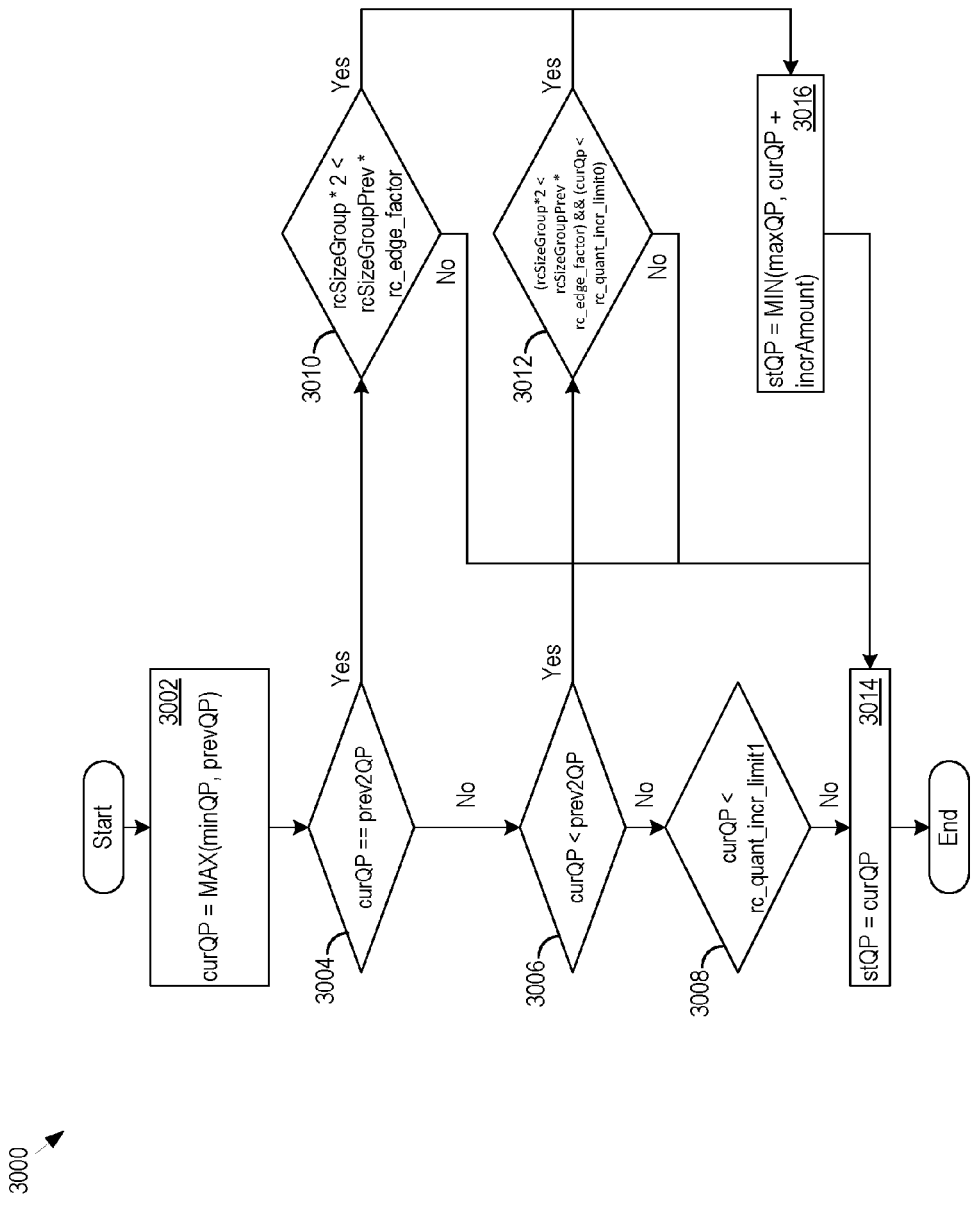

3000
Start
3002
curQP = MAX(minQP, prevQP)
curQP == prev2QP
3004
Yes
No
curQP < prev2QP
3006
Yes
No
curQP < rc_quant_incr_limit1
3008
No
stQP = curQP
3014
End
rcSizeGroup * 2 < rcSizeGroupPrev * rc_edge_factor
3010
Yes
No
(rcSizeGroup*2 < rcSizeGroupPrev * rc_edge_factor) && (curQp < rc_quant_incr_limit0)
3012
Yes
No
stQP = MIN(maxQP, curQP + incrAmount)
3016

Figure 30

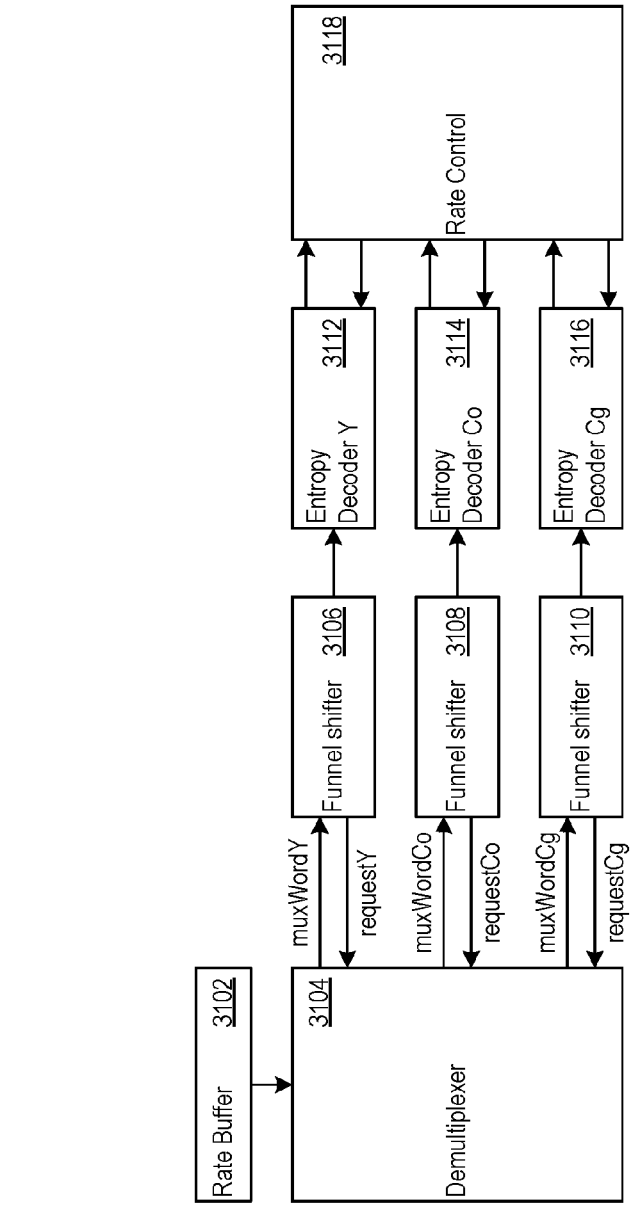

3118
Rate Control
3112
Entropy Decoder Y
3114
Entropy Decoder Co
3116
Entropy Decoder Cg
3106
Funnel shifter
3108
Funnel shifter
3110
Funnel shifter
muxWordY
requestY
muxWordCo
requestCo
muxWordCg
requestCg
3102
Rate Buffer
3104
Demultiplexer
3100

Figure 31

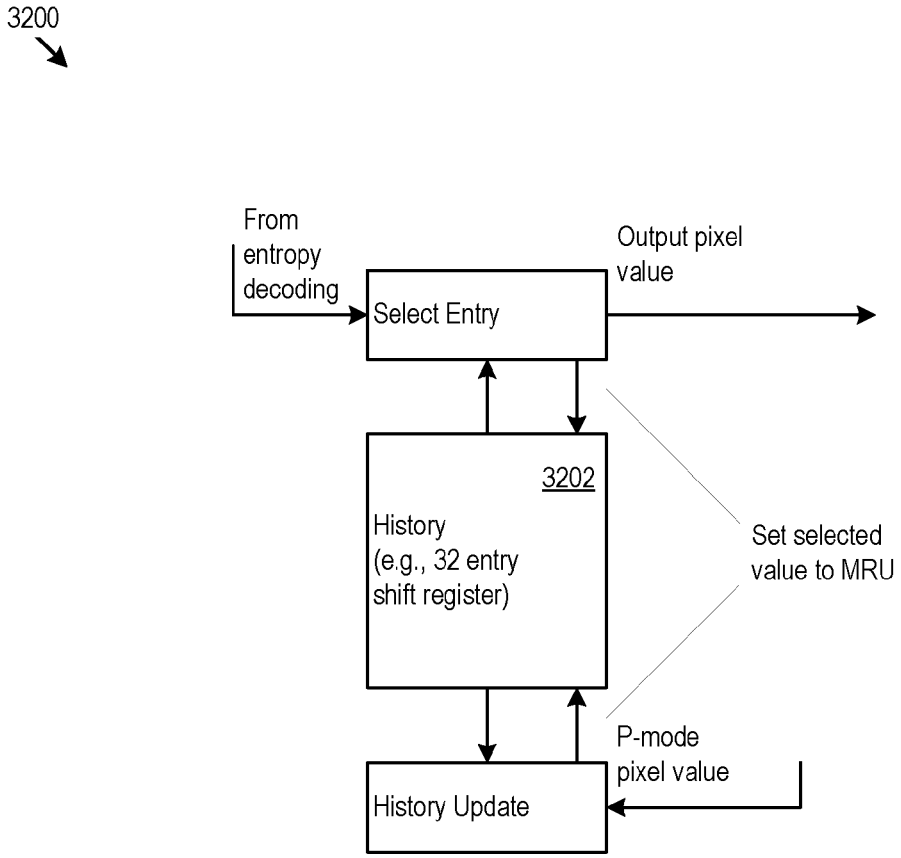

3200
From entropy decoding
Select Entry
Output pixel value
3202
History (e.g., 32 entry shift register)
Set selected value to MRU
P-mode pixel value
History Update

Figure 32

BOUNDED RATE COMPRESSION WITH RATE CONTROL FOR SLICES

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority to provisional application Ser. No. 61/709,316, filed 3 Oct. 2012 and provisional application Ser. No. 61/764,807, filed 14 Feb. 2013, which are entirely incorporated herein by reference. This application also claims priority to provisional application Ser. No. 61/764,891, filed 14 Feb. 2013, provisional application Ser. No. 61/770,979, filed 28 Feb. 2013, provisional application Ser. No. 61/810,126, filed 9 Apr. 2013, provisional application Ser. No. 61/820,967, filed 8 May 2013 provisional application Ser. No. 61/832,547, filed 7 Jun. 2013, and provisional application Ser. No. 61/856,302, filed 19 Jul. 2013.

TECHNICAL FIELD

This disclosure relates to image processing. This disclosure also relates to compression and decompression techniques for image transmission and display.

BACKGROUND

Immense customer demand has driven rapid advances in display technologies, image analysis algorithms, and communication technologies, as well as the widespread adoption of sophisticated image display devices. As just a few examples, these devices range from DVD and Blu-ray players that drive high resolution displays for home theaters, to the now ubiquitous smart phones and tablet computers that also have very high resolution displays. Improvements in image processing techniques will continue to expand the capabilities of these devices.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows an example architecture in which a source communicates encoded data to a sink.

FIG. **2** is an example of an encoder.

FIG. **3** shows a parallel processing architecture.

FIG. **4** shows an example of a predictor and quantizer.

FIG. **5** shows example sample locations.

FIG. **6** shows examples of a coded format for compressed samples.

FIG. **7** shows an example of a virtual buffer model.

FIG. **8** shows an example decoder.

FIG. **9** shows example logic for encoding.

FIG. **10** shows example logic for decoding.

FIG. **11** shows an example encoding and decoding system.

FIG. **12** shows an example of a picture and a picture parameter set.

FIG. **13** shows another example of an encoder.

FIG. **14** shows another example of a decoder.

FIG. **15** illustrates samples sets for block search.

FIG. **16** illustrates an example of indexed color history.

FIG. **17** shows an example of a portion of a slice using substream multiplexing.

FIG. **18** shows an example of substream demultiplexing logic.

FIG. **19** shows an example of substream multiplexing logic.

FIG. **20** shows an example of slice timing and delays.

FIG. **21** shows an example of 3x1 partial SADs that form 9x1 SAD.

FIG. **22** shows an example of original pixels used for encoder flatness checks.

FIG. **23** shows an example of encoder logic.

FIG. **24** shows an example of a buffer level tracker.

FIG. **25** shows an example of encoder logic that may implement rate control for slices.

FIG. **26** shows an example of bit trajectories over time.

FIG. **27** shows examples of offset fullness and actual fullness.

FIG. **28** shows example threshold ranges.

FIG. **29** shows an example of short term rate control.

FIG. **30** shows an example of QP increment logic.

FIG. **31** shows an example of substream demultiplexing in a decoder.

FIG. **32** shows indexed color history logic in a decoder.

DETAILED DESCRIPTION

FIG. **1** shows an example architecture **100** in which a source **150** communicates with a sink **152** through a communication link **154**. The source **150** or sink **152** may be present in any device that manipulates image data, such as a DVD or Blu-ray player, a smartphone, a tablet computer, or any other device. The source **150** may include an encoder **104** that maintains a virtual buffer **114**. The sink **152** may include a decoder **106**, memory **108**, and display **110**. The encoder **104** receives source data **112** (e.g., source image data) and may maintain the virtual buffer **114** of predetermined capacity to model or simulate a physical buffer that temporarily stores compressed output data. The encoder **104** may also evaluate the encoded symbols for transmission at a predetermined bit rate. The encoder **104** may specify the bit rate, as just two examples, in units of bits per pixel, or in units of bits per unit of time.

The encoder **104** may determine the bit rate, for example, by maintaining a cumulative count of the number of bits that are used for encoding minus the number of bits that are output. While the encoder **104** may use a virtual buffer **114** to model the buffering of data prior to transmission of the encoded data **116** to the memory **108**, the predetermined capacity of the virtual buffer and the output bit rate do not necessarily have to be equal to the actual capacity of any buffer in the encoder or the actual output bit rate. Further, the encoder **104** may adjust a quantization step for encoding responsive to the fullness or emptiness of the virtual buffer. An exemplary encoder **104** and operation of the encoder **104** are described below.

The decoder **106** may obtain the encoded data **116** from the memory **108**. Further, the decoder **106** may determine the predetermined virtual buffer capacity and bit rate, and may determine the quantization step that the encoder **104** employed for encoding the encoded data **114**. As the decoder **106** decodes the encoded data **116**, the decoder **106** may also determine the fullness or emptiness of the virtual buffer **114** and adjust the quantization step used for decoding. That is, the decoder **106** may track the operation of the encoder **104** and determine the quantization step that the encoder **104** used. The decoder **106** decodes the encoded data **116** and provides video data **118** to a display **110**. In some implementations, the quantization step is not present in the encoded data **116**, saving significant bandwidth. Examples of decoders **106** and encoders **104**, and their operation are described below.

The memory **108** may be implemented as Static Random Access Memory (SRAM), Dynamic RAM (DRAM), a solid

state drive (SSD), hard disk, or other type of memory. The display link **154** may be a wireless or wired connection, or combinations of wired and wireless connections. The encoder **104**, decoder **106**, memory **108**, and display **110** may all be present in a single device (e.g. a smartphone). Alternatively, any subset of the encoder **104**, decoder **106**, memory **108**, and display **110** may be present in a given device. For example, a Blu-ray player may include the decoder **106** and memory **108**, and the display **110** may be a separate display in communication with the Blu-ray player.

FIG. **2** shows an example of an encoder **200**. The encoder **200** encodes the video data **202**. The video data **202** may take the form of a series of successive frames **202-0**, . . . , **202-***x*, for example. The frames **202-0**, . . . , **202-***x* may take the form of 2-dimensional matrices of pixel components, which may be represented in any color space such as the Red/Green/Blue (RGB), YUV, Luminance Y/Chroma Blue Cb/Chroma Red Cr (YCbCr), Luminance Y/Chroma Orange/Chroma Green (YCoCg), Alpha, Red, Green, Blue (ARGB), or other color space. Each of the pixel components may correspond to a spatial location. While the matrices may be overlaid to form a picture, each of the pixel components in the matrices are not necessarily co-located with pixel components in other matrices.

Each pixel component may be encoded with a value comprising a predetermined number of bits, such as eight, ten, or twelve bits per pixel component. The encoding may employ, as examples, 10 bit YCbCr 4:2:2, 8 bit YCbCr 4:2:2, 10 bit YCbCr 4:4:4, 8 bit YCbCr 4:4:4, 8 bit ARGB 32, or 8 bit RGB 24 encoding. The encoder **200** may receive the pixel components of the frames in raster scan order: left to right, top to bottom. In certain implementations, the video encoder **200** may receive the pixel components at a predetermined rate. The predetermined rate may correspond to the real-time frames per second display rate.

The video encoder **200** may include an input, predictor & quantizer **204**, a mapping and variable length coder (VLC) **206**, rate controller **208**, a rate buffer **210**, and memory (e.g., DRAM) **212**. The video encoder **200** receives and encodes the pixel components. While the number of bits representing pixel components coming into the video encoder **200** may be constant (per pixel component), the number of bits representing each coded pixel may vary dramatically. The encoder **200** may increase the number of bits representing coded pixels by reducing the quantization step, or decrease the number of bits by increasing the quantization step.

The input, predictor & quantizer **204** predicts and quantizes the pixel components, resulting in quantized residuals. In certain implementations, the input, predictor, & quantizer **204** may predict a pixel component from previously encoded and reconstructed pixel components in the same frame, e.g., **202-0**. The mapper and variable length coder **206** codes the quantized residuals, resulting in coded bits.

The input, predictor & quantizer **204** may use a predetermined initial quantization step for quantizing a predetermined amount of data, such as video pixel data. The mapping and variable length coder **206** signals the rate controller **208**, which in turn instructs the input, predictor & quantizer **204** to increment, decrement, or leave unchanged the quantization parameter, as will be described in more detail below.

The mapping and variable length coder **206** may code the quantized sample values using their natural 2's complement binary values. The number of bits that the mapping and variable length coder **206** uses to code each value may be determined dynamically by a combination of recent history of coded values of the same pixel component and a prefix value associated with each unit of samples.

The rate controller **208** determines whether to increment, decrement, or leave unchanged the quantization step. The rate controller **208** may perform the quantization step analysis, e.g., by simulating or modeling a buffer of predetermined capacity that it evaluates at a predetermined bit rate. The modeled buffer may be referred to as a virtual buffer. If the virtual buffer is becoming full, the rate controller **208** may increase or increment the quantization step. If the virtual buffer is becoming empty, the rate controller **2098** may decrease or decrement the quantization step. Further aspects of this are described below with respect to rate control for slices.

The rate controller **208** may determine the fullness of the virtual buffer by, e.g., counting the bits that are used to encode the input received over a given number of input samples and subtracting the product of the predetermined bit rate, in bits per sample, and the number of input samples. The number of input samples may be as few as one sample.

A decoder may decode the encoded data starting with the initial quantization step. As the decoder decodes the encoded data, the decoder may also determine the fullness of the virtual buffer. The decoder may determine the fullness or emptiness by observing the amount of bits that were used to encode an amount of decoded data corresponding to the number of input samples. The decoder may then determine the quantization step decision that was made at the encoder **200**. Accordingly, the encoder **200** does not need to explicitly transmit the quantization step to the rate controller or any other logic in the decoder.

FIG. **3** shows a parallel processing architecture **300**. The demultiplexer **302** receives the input pixel components **304**, and separates each pixel component into constituent parts, e.g., Alpha **306**, Red **308**, Green **310**, and Blue **312**. The prediction & quantization blocks **314**, **316**, **318**, and **320** are associated with a particular one of the constituent parts of the pixel components. There may be any number of such blocks and they may operate in parallel. In the case of a format with four pixel components, such as ARGB, each prediction & quantization block processes a particular component part. When the architecture **300** processes pixel components with fewer constituent parts than prediction & quantization blocks, then some of the prediction & quantization blocks need not operate for the processing of those pixel components. The prediction & quantization blocks **314-320** may provide quantized residuals to a particular one of component mappers **322**, **324**, **326**, and **328**. The component mappers **322-328** may also operate in parallel.

The mappers **322-328** provide mapped quantized residuals 'E' to a multiplexer **330**. The multiplexer **330** multiplexes the mapped quantized residuals 'E' into a residual stream **332** that is provided to the variable length coder **334**. Alternatively, there may be a variable length encoder associated with each component mapper, and the multiplexer **330** may multiplex the variable length encoded quantized residuals output by the multiple variable length encoders.

FIG. **4** shows an example of a predictor and quantizer **400**. The predictor and quantizer **400** includes a buffer **402**, first delay logic **404** (implementing, e.g., six sample delay), a prediction engine **406**, and second delay logic **408** (implementing, e.g., 3 sample delay). The buffer **402** may store the previous reconstructed image line. The prediction engine **406** receives the current pixel component 'x', reconstructed pixel components 'w', 't', 's', 'g', 'c', 'b', 'd', and 'h' from the previous line from the first delay blocks **404**, and reconstructed pixels from the left on the current line, 'k', 'r', 'f', 'a' from the second delay blocks **408**.

In order to use reconstructed pixel components, instead of the input pixel components, the quantizer **410** may provide quantized residuals E' to an inverse quantizer **412**. The inverse quantizer **412** inverse quantizes the quantized residuals. The reconstructed pixels 'Rx' are generated from the quantized residuals E' and the predicted values from the prediction engine.

The prediction engine **406** may include an Edge prediction engine **414**, LS prediction engine **416**, Left prediction engine **418**, and ABCD prediction engine **420**. As described above, the prediction engine **406** predicts the current pixel component 'x' from reconstructed pixel components 'w', 't', 's', 'g', 'c', 'b', 'd', and 'h' from the previous line, and reconstructed pixels from the left on the current line, 'k', 'r', 'f', 'a', thereby resulting in a residual E" representing the pixel component 'x'.

The operation of the prediction engine **406** will now be described with reference to FIG. **5**, which shows example pixel components **500**. The prediction engine **406** may adaptively predict pixel components from neighboring reconstructed pixels of the line above, and the left pixels of the same line of the pixel to be predicted. For example, the prediction engine **406** may predict pixel 'x' from a combination of any of the reconstructed pixels 't', 's', 'g', 'c', 'b', 'd', 'h', 'k', 'r', 'f', and 'a'.

The spatial prediction adaptively chooses an output from one of the four candidate prediction engines: the Edge prediction engine **414**, LS prediction engine **416**, Left prediction engine **418**, and ABCD prediction engine **420** as its predictor for the current pixel component. The choice may be made according to the prediction errors determined for one or more previous reconstructed pixel components, considering the candidate predictors. This operation may be the same in both the encoder and decoder, and no prediction control information needs to be included in the encoded data. The decoder may implement an identical prediction mode algorithm and deduce the prediction mode used by the encoder. Once a predictor is selected, the value of each sample is predicted using the selected predictor. The residual value E" is calculated as the difference between the predicted value and the actual sample value.

LS Prediction Engine **416**

The LS prediction engine **416** may produce a predicted value Px of the current sample 'x' according to the following:

```
if (c >= max(a, b))
        Px = min(a, b);
else {
        if (c <= min(a, b))
                Px = max(a, b);
        else Px = a + b − c;}
```

ABCD Prediction Engine **420**.

The ABCD prediction engine **420** may produce the prediction value Px=(a+b+c+d+2)/4. This is an average of four neighboring samples.

Left Prediction Engine **418**

The Left prediction engine **418** may use the reconstructed value of the left pixel of the current sample as its prediction value. In other words, Px='a'.

Edge Prediction Engine **414**

The Edge prediction engine **414** may employ more neighboring pixels than the LS prediction engine **416**. The Edge prediction engine **414** may detect an edge at several possible angles around the current sample 'x', and use the edge information in the prediction. The Edge prediction engine **414** may search, as examples, for directions of an edge that are horizontal, vertical, 45 degree, 135 degree, about 22.5 degrees and about 157.5 degrees. The Edge prediction engine **414** may be implemented in two stages. The first stage is edge detection. The second stage is edge selection.

Some options may be specified for the prediction function. The use of the reconstructed sample value 'a', which is immediately to the left of 'x', may be disabled by configuring the Edge prediction engine **414** with a parameter such as NOLEFT=1. Avoiding the use of sample 'a' may allow more time for the prediction, quantization and inverse quantization path to function, which may be an advantage in high throughput systems where circuit timing may make it difficult to reconstruct sample 'a' quickly. The use of the reconstructed sample values 'a' and 'f', which are the two samples immediately to the left of 'x', may be disabled by configuring the Edge prediction engine **414** with a parameter such as NOLEFT=2 (also referred to as NO2LEFT). This allows even more time for the prediction, quantization and inverse quantization path to function. When circuit timing needs three clock cycles for prediction, quantization and inverse quantization, the use of NOLEFT=2 facilitates a throughput of one sample per clock.

The individual prediction engines from the four listed above may be selectively enabled and disabled. For certain classes of content, better performance may be obtained by utilizing a subset of the prediction functions. When predicting samples along the top and left edges of an image, for example, the Left prediction engine **418** may be employed, as specified below.

NOLEFT=1 Option

When NOLEFT=1, the reconstructed sample value 'a' in the LS prediction engine **416**, ABCD prediction engine **418**, and Edge prediction engine **420** is replaced by its prediction Pa using the reconstructed samples 'f', 'g', and 'c' according to the following:

```
if (ABS(g−c) > ABS(g−f)*3)
        Pa = c;
else {
        if (ABS(g−f) > ABS(g−c)*3)
                Pa = f;
        else Pa = (f+c+1)/2;}
```

NOLEFT=2 Option

When NOLEFT=2, the reconstructed sample values 'f' and 'a' in the LS prediction engine **416**, ABCD prediction engine **418**, and Edge prediction engine **420** are replaced by their predictions Pf and Pa using the reconstructed samples 'r', 's', 'g', and 'c'. The prediction of 'a' may use the same approach as in NOLEFT, except that 'f' is replaced by Pf according to the following:

$$Pf=(r+g+s+c+2)/4;$$

Edge prediction engine with NOLEFT=0, NOLEFT=1, NOLEFT=2

When NOLEFT=0, the left sample is used in the prediction, and the following may be applied to the edge detection:

```
    if ( (2*ABS(a−c) > 6*ABS(c−b)) && 2*ABS(a−c) > 6*ABS(c−g)
&& 2*ABS(a−c) > 6*ABS(a−f) )
    {
        edge1 = 0;
        strength1 = ABS(c−b);
    }
```

-continued

```
else if ( 2*ABS(b-c) > 6*ABS(c-a) && 2*ABS(c-d) >
6*ABS(c-a) )
{
    edge1 = 1;
    strength1 = ABS(c-a) ;
}
else
{
    strength1 = max_strength;
    edge1 = 7;
}
if ( 2* ABS(a-g) > 6*ABS(a-b) && 2* ABS(a-g) > 6*ABS(f-c) )
{
    edge2 = 2;
    strength2 = ABS(a-b);
}
else if( 2* ABS(a-b) > 6*ABS(a-g) && 2* ABS(a-b) >
6*ABS(s-f) )
{
    edge2 = 3;
    strength2 = ABS(a-g) ;
}
else
{
    strength2 = max_strength;
    edge2 = 7;
}
if ( 2*ABS(a-g) > 6*ABS(a-d) )
{
    edge3 = 4;
    strength3 = ABS(a-d) ;
}
else if ( 2*ABS(a-b) > 6*ABS(a-s) )
{
    edge3 = 5;
    strength3 = ABS(a-s) ;
}
else
{
    strength3 = max_strength;
    edge3 = 7;
}
```

When NOLEFT=1, the left sample is not used in the prediction, and the following may be applied to the edge detection:

```
if ( (2*ABS(f-g) > 6*ABS(c-g)) && 2*ABS(f-g) > 6*ABS(s-g)
&& 2*ABS(f-g) > 6*ABS(r-f) )
{
    edge1 = 0;
    strength1 = ABS(c-g);
}
else if ( 2*ABS(g-c) > 6*ABS(f-g) && 2*ABS(b-g) >
6*ABS(g-f) )
{
    edge1 = 1;
    strength1 = ABS(f-g);
}
else
{
    strength1 = max_strength;
    edge1 = 7;
}
if ( 2* ABS(f-s) > 6*ABS(f-c) && 2* ABS(f-s) > 6*ABS(r-g) )
{
    edge2 = 2;
    strength2 = ABS(f-c);
}
else if ( 2* ABS(f-c) > 6*ABS(s-f) && 2* ABS(f-c) >
6*ABS(r-t) )
{
    edge2 = 3;
    strength2 = ABS(s-f);
}
else
{
    strength2 = max_strength;
    edge2 = 7;
}
if ( 2*ABS(s-f) > 6*ABS(f-b) )
{
    edge3 = 4;
    strength3 = ABS(f-b);
}
else if ( 2*ABS(f-c) > 6*ABS(f-t) )
{
    edge3 = 5;
    strength3 = ABS(f-t);
}
else
{
    strength3 = max_strength;
    edge3 = 7;
}
```

When NOLEFT=2, the two left samples are not used in the prediction, and the following may be applied to the edge detection:

```
if ( (2*ABS(r-s) > 6*ABS(g-s)) && 2*ABS(r-s) > 6*ABS(t-s)
&& 2*ABS(r-s) > 6*ABS(k-r) )
{
    edge1 = 0;
    strength1 = ABS(g-s);
}
else if ( 2*ABS(s-g) > 6*ABS(r-s) && 2*ABS(c-s) >
6*ABS(s-r) )
{
    edge1 = 1;
    strength1 = ABS(r-s);
}
else
{
    strength1 = max_strength;
    edge1 = 7;
}
if ( 2* ABS(r-t) > 6*ABS(r-g) && 2* ABS(r-t) > 6*ABS(k-s) )
{
    edge2 = 2;
    strength2 = ABS(r-g);
}
else if ( 2* ABS(r-g) > 6*ABS(t-r) && 2* ABS(r-g) >
6*ABS(k-w) )
{
    edge2 = 3;
    strength2 = ABS(t-r);
}
else
{
    strength2 = max_strength;
    edge2 = 7;
}
if ( 2*ABS(t-r) > 6*ABS(r-c) )
{
    edge3 = 4;
    strength3 = ABS(r-c);
}
else if ( 2*ABS(r-g) > 6*ABS(r-w) )
{
    edge3 = 5;
    strength3 = ABS(r-w);
}
else
{
    strength3 = max_strength;
    edge3 = 7;
}
```

The parameter 'max_strength' may be defined as the largest possible absolute difference between two samples. This parameter may be related to the pixel data format, e.g., for 8-bit data, max_strength=255, for 10-bit data, max_

strength=1023. The same edge selection logic may be applied in each case of NOLEFT=0, NOLEFT=1 and NOLEFT=2, except that the sample value 'a' may be replaced by its prediction Pa when NOLEFT=1 or NOLEFT=2, and the sample value 'f' may be replaced by its prediction Pf when NOLEFT=2:

```
if (strength1 <= strength2)
{
        if (strength1 <= strength3)
        {
                edge = edge1;
                strength = strength1;
        }
        else
        {
                edge = edge3;
                strength = strength3;
        }
}
else
{
        if (strength2 <= strength3)
        {
                edge = edge2;
                strength = strength2;
        }
        else
        {
                edge = edge3;
                strength = strength3;
        }
}
if (strength == max_strength || edge == 7)
        Px = (a+c+b+d+2) / 4;
else
{
        switch(edge)
        {
        case 0: Px = a;
        case 1: Px = b;
        case 2: Px = d;
        case 3: Px = c;
        case 4: Px = h;
        case 5: Px = g;
        }
}
```

Predictor Selection

A Unit may be considered to be a logical grouping of adjacent samples of the same component. For example, the Unit size may be selected to be equal to two. A Unit size may be the number of samples comprised by a Unit. In alternative implementations, the Unit size may be selected to have a value of one, three, four or another value. In one embodiment, when the Unit size is selected to be equal to two, for every pair of samples of one component, a selected set (up to all) of the candidate predictors may be evaluated using the previous pair of samples of the same component, and the predictor that performs best for that previous pair is selected for the current pair. The selection of a predictor may be made on boundaries that do not align with Units. There may be certain exceptions under which the set of candidate predictors is restricted, for example when samples to the left or above are not available, or for example when one or more predictors are not enabled.

For the first pair of samples of the image, e.g., the two samples on the left edge of the top line, the Left prediction engine **418** may be selected as the predictor. Further, for the first pair of samples of each line other than the first, the LS prediction engine **418** may be selected. Sample values that are not available for use in prediction may be assigned a pre-determined value, for example one half of the maximum range of sample values.

For other pairs of samples, the predictor may be selected according to the estimated prediction errors of the left pair of samples, which may be calculated by all four predictors. When the reconstructed value of the current sample 'x' is found, the estimated prediction error for the current sample can be calculated as follows:

$$err\_sample=ABS(x'-Px)$$

In the above equation, Px is the predicted value of the current sample from each of the four predictors. The prediction error of one predictor is the sum of err_sample over both samples in a pair of samples for a predictor. The predictor with the smallest prediction error is then selected as the predictor for the next pair of samples of the same component.

Note when NOLEFT=1, the prediction error of the left sample is not available. Assuming the current sample is 'x' in FIG. **5**, then if NOLEFT=0, the prediction engine selected by the left pair, the samples of 'f' and 'a', is used for the current sample pair. If NOLEFT=1, the predictor selected by the smallest prediction error of the available left pair may be used, e.g., the samples of 'r' and 'f' if 'x' is the second sample of the pair, or samples of 'r' and 'k' is 'x' is the first sample of the pair. If NOLEFT=2, the predictor selected by the smallest prediction error of the samples of 'r' and 'k' may be used if 'x' is the first sample of the pair, or samples of 'k' and its immediately left one if 'x' is the second sample of the pair. The residual or error value E" may be determined as: E"=x−Px.

The reconstructed sample value of 'x', for use in future predictions, may be obtained as follows:

$$x'=Px+E''*\text{QuantDivisor};$$

$$\text{if } (x'<0)x'=0;$$

$$\text{else if } (x'>\text{MAXVAL})x'=\text{MAXVAL};$$

The value QuantDivisor is defined below. MAXVAL is the maximum value that can be coded by the uncompressed video sample word size, e.g., 1023 for 10 bit video, and 255 for 8 bit video. In one implementation, Cb and Cr are non-negative integers.

The operation of the mapper and variable length coder **206** is described with reference to FIG. **6**, which shows examples of sample units **600**, which are also referred to as Units. The mapper and variable length coder **206** may use entropy coding to code sample values using their natural 2's complement binary values. The number of bits used to code each value may be determined dynamically by a combination of the recent history of coded values of the same component and a prefix value associated with each Unit **605** of samples. In certain implementations, a Unit **605** comprises two samples **610** of a particular component type, e.g., Y, Cb or Cr, or Alpha, R, G or B. In some implementations, the Cb and Cr samples are coded together in one Unit. The same set of components may be used for the prediction of the number of bits.

Each Unit **605** of samples has a Unit sample size. A Unit sample size may be the size in bits of each of the samples in a Unit. The Unit **605** sample size may be large enough to code each of the samples contained in the Unit **505**, and it may be larger. The size of one sample may be the number of bits used to code the sample's value in 2's complement. For example, a value of 0 has a size of 0, a value of −1 has a size of 1, a value of −2 or 1 has a size of 2, a value of −4, −3, 2 or 3 has a size of 3, and so on.

A Unit **605**, may have a maximum sample size, which is the maximum of the sizes of all the samples in the Unit **605**. A Unit **605** may also have a predicted size. In one implementation, if the predicted size is greater than or equal to the maximum sample size, then the Unit **605** sample size is equal to the predicted size. In one implementation, if the maximum sample size is greater than the predicted size, then the difference, which is always non-negative, is coded in the prefix value **612**, and the maximum sample size may be used as the Unit **605** sample size. In another implementation, if the maximum sample size is different from the predicted size, then the difference, which may be positive or negative, is coded in the prefix value **612**. The prefix value may use unary coding, e.g., for implementations with non-negative prefix values, the value 0 has the code 1 (binary), the value 1 has the code 01, the value 2 has the code 001, and so on. The Unit sample size is the sum of the predicted size and the prefix value **612**. For 10 bit video, the greatest possible sample size is 10, and the smallest possible predicted size is 0, so the greatest possible prefix value is 10, which occupies 11 bits i.e. 0000 0000 001. For implementations with signed prefix values, signed prefix values may be unary coded.

The predicted size may be a function of the sizes of previously coded samples. In one implementation, the predicted size is the average, with rounding, of the sizes of the samples of the same component of the previous two samples, e.g., of the previous Unit, given that the Unit size is 2. If the Unit size is 4, the predicted size may be the average of the sizes of the four samples of the same component of the previous Unit. If the Unit size is 3, the predicted size may be generated by the average of the sizes of the last two samples of the same component of the previous Unit, thereby avoiding division by 3. Alternatively, if the Unit size is 3, the predicted size may be generated as a weighted sum of 3 samples of the previous unit of the same component. The weights may be, for example, (¼, ¼, ½).

For example, if a component of an image, after quantization, is such that the size of the samples is 2 for many consecutive samples, then the predicted size is 2, and the prefix value is 0. Therefore the prefix code is '1', and each sample is coded using 2 bits, and a Unit of two samples has a total of 5 bits. In the event of a transient causing a sudden increase in the sample size, the prefix value codes the increase in the sizes. In the event of another transient causing a sudden decrease in the sample size, the prefix value may be 0 and the Unit sample size may be equal to the predicted size, which may be in excess of the sizes of the samples in the Unit. Therefore each sample may be coded with a number of bits equal to the predicted size, even though their own sizes are less. Following a transient, in the absence of another change in sample sizes, the Unit sample size and predicted size converge again. This technique results in very efficient coding of samples, given that the sizes of the samples may change from Unit to Unit, particularly when the sizes do not frequently change very rapidly.

The delta size Unit variable length coding (DSU-VLC) scheme facilitates efficient encoding and decoding at high speed in hardware, in part because it does not rely upon VLC tables. The number of bits in a Unit to be decoded is determined from the prefix value (counting zeros) and the predicted size, which can be determined before encoding or decoding the current Unit. It is feasible to encode or decode one Unit per clock, and faster decoding approaches are also feasible. Encoding can encode multiple Units in parallel, for greater throughput. The Unit size may be selected to be greater than two for various reasons. For example, larger Unit size may be chosen where the usage imposes a throughput requirement that cannot practically be met with a Unit size of 2, in which case a Unit size of 3 or 4 may be used.

Referring again to FIG. **4**, the quantizer **410** quantizes the residuals E", which in general includes the case of lossless coding, using a quantization parameter Quant. Quant may take on values ranging from 0, signifying lossless, to the value that corresponds to the highest value of Quant Divisor[ ] (see below). With an exemplary set of values of QuantDivisor and QuantOffset shown below, the value of Quant ranges from 0 to 17.

The quantizer **410** may perform quantization on the residual value E" as follows:

```
if (Quant = 0)
        E' = E";
else
        if (E" >= 0)
                E' = (E" + QuantOffset[Quant]) / QuantDivisor[Quant];
        else E' = (E" – QuantOffset[Quant]) / QuantDivisor[Quant];
```

where division may be with truncation, as, e.g., in the 'C' language.

The set of divisors may be:

int QuantDivisor[ ]={1, 3, 5, 7, 9, 10, 12, 14, 16, 18, 20, 24, 28, 32, 48, 64, 128, 256};

The associated set of offsets, the rounding constants, may be:

int QuantOffset[ ]={0, 1, 2, 3, 4, 4, 5, 6, 7, 8, 9, 11, 13, 15, 23, 31, 63, 127};

In this approach, there are 4 odd-valued divisors (3, 5, 7 and 9), and seven that are products of one of these odd-valued divisors and one of five other values, each of which is a power of 2: 2**N. As a result, in one implementation, the quantization function supports 4 odd-valued divisors.

The use of this particular set of values of Quant Divisor[ ] provides good compression with low complexity. Note that division by the odd numbers can be performed in hardware using multiplication by one of a small set of optimized constant values.

In other implementations, the divisors may be selected such that they do not have odd factors. For example:

int QuantDivisor[ ]={1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096};

int QuantOffset[ ]={0, 0, 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047};

Rate Control

The value of Quant is determined via a rate control technique, which may be performed identically in both the encoder and decoder. The rate control technique may base its decisions on a measure of the activity of the most recently coded predetermined number of pixel components and on the fullness of the buffer model. The predetermined number may be, for example, 3, 2, or some other number. The value of Quant may be updated once per coded predetermined number of pixel components.

FIG. **7** shows an example of a virtual buffer model **700**. The virtual buffer model **700** is in communication with a bitstream source **702**, the rate controller **208**, and a bitstream consumer **706**. The virtual buffer model **700** models the behavior of a rate buffer where the output bit rate is a specified bit rate. The specified bit rate may be in units of bits per pixel or per group of pixels, or it may be in other units such as bits per unit of time, such as bits per second. The bitstream consumer **706** may model the consumption of bits at a specified rate. The bitstream source **702** may be the output of the mapper and variable length coder **206**, for

example. A group of pixels may comprise a predetermined number of pixels, for example two, three, four, or some other number.

Bits enter the virtual buffer model **700** when they are created. For example, the number of bits used to code a Group is added to the model **700** when the Group is coded. Bits leave the virtual buffer model **700** according to a pre-determined schedule. For example, the schedule may specify a constant rate in units of bits per group. The virtual buffer model **700** may be implemented as an accumulator **708**, in which one value is added and other value is subtracted per Group. Alternatively, the schedule of removing bits from the virtual buffer model **700** may be in units of bits per second. Alternatively, the times at which bits are added to or subtracted from the buffer model **700** may be finer or coarser than a Group, and may use a construct other than a Group, such as a sample, a macroblock, a slice or a picture. In order to model the behavior of a First In First Out (FIFO) buffer, the fullness of the virtual buffer model **700** may be clamped to 0 when subtracting a number of bits from the fullness that would otherwise result in a negative value of fullness.

When the output bit rate used in the virtual buffer model **700** is less than or equal to the actual bit rate at which bits are removed from the rate buffer in an encoder, and the rate controller **704** ensures that the virtual buffer model **700** does not overflow, the rate buffer also does not overflow. More generally, the encoder may use the virtual buffer model **700** to manage the rate of creation of bits by the encoder such that another virtual buffer model, which may be applied later to the encoder's bit stream, does not overflow or underflow. The bit rate at which bits leave the virtual buffer model can be changed at any time to any supported value. If the actual rate at which bits leave the rate buffer equals or approximates the rate at which bits leave the virtual buffer model, the encoder's bit rate can be set to any supported bit rate with effectively instantaneous response. Because the rate control uses the virtual buffer model to manage the rate of creation of bits, the rate control function does not need to monitor the rate at which bits leave the rate buffer.

In one implementation, the encoder and decoder perform identical rate control (RC) decisions, which control the value of the quantizer, or Quant, without the encoder transmitting any bits that specifically indicate quantization control. The rate control may depend on the activity, measured by the sizes of the samples, of the previous Group, as well as fullness of the virtual buffer model, and a measure of the strength of an edge, if any, in the preceding samples. The rate control may use several configurable thresholds. Units **605** are organized into Groups **710**. Groups **710** are utilized to organize the samples to facilitate the buffer model and rate control. In another exemplary implementation, the decoder does not perform the same rate control decisions as the encoder, and the encoder transmits bits which indicate at least a portion of the quantization control.

In one implementation, the encoder, including the rate controller **208**, ensures that the virtual buffer model **700** never exceeds a defined maximum fullness, while choosing quantization levels to maximize overall subjective image quality. For some images and bit rates, both may be achieved relatively easily, while for others, the buffer fullness may vary and approach or reach the size of the virtual buffer model **700** at times and the quantization may vary and may reach the maximum allowed value at times.

The virtual buffer model **700** may represent a FIFO of predetermined size, BufferSize. The value of BufferSize may be chosen according to the particular application. A larger size generally facilitates better compression for a given bit rate and image contents, and vice versa. A larger size also implies a larger amount of space available in a physical rate buffer, as well as potentially increased latency. In an exemplary implementation, at the start of a picture, the buffer model **700** is initialized to be empty. Alternatively, the virtual buffer model **700** fullness may be retained from one picture to the next, or it may be initialized to some other value.

As each Group **710** of samples is encoded, the number of bits used to code the Group is added to the accumulator in the virtual buffer model **700**. After each Group is coded, a number equal to the budget of bits per Group, e.g., the specified bit rate, is subtracted from the accumulator, with the result clamped to 0 to enforce non-negative fullness. In implementations where the decoder mimics the rate control of the encoder, the same operation happens in the decoder: as each Group is decoded, the number of bits that the Group occupies is added to the model and the specified bit rate, e.g., the budget number of bits per Group, is subtracted, with the result clamped to 0. This way the encoder and decoder buffer models track exactly for every Group in each picture. The rate controller **208** can guarantee that the buffer fullness never exceeds the defined maximum value, e.g., the buffer size, by adjusting the value of Quant.

In one implementation, at the start of each picture, the quantization value Quant is initialized to 0, corresponding to lossless coding. In another implementation, the value of Quant is initialized to a non-zero value. The value of Quant may be adjusted dynamically to avoid overflowing the buffer model while maximizing the compressed image quality. The rate control algorithm may facilitate encoding of difficult images at low bit rates with minimum visible quantization errors, as well as encoding difficult images at higher bit rates with no visible quantization error.

In one implementation, the activity level of each Group is measured. The activity level may be the maximum quantized residual size of each Unit in the Group, times the number of samples in a Unit (e.g., either 2, 3, or 4), plus 1 (corresponding to a prefix value of 0), summed over all of the Units in the Group. The quantized residual sizes are after quantization using the current value of Quant. As an example of 2 samples per unit and 3 units per group, the numbers of bits for sample 0 and 1 are SampleSize[0] and SampleSize[1] respectively. Assume the maximum of the two samples for unit 0 is MaxSizeUnit[0]=MAX(SampleSize[0], SampleSize[1]), then the activity level for the group is RcSizeGroup=MaxSizeUnit[0]+1+MaxSizeUnit[1]+1+MaxSizeUnit[2]+1. Another parameter that calculates the real number of bits coded in the last Group, e.g., BitsCodedCur, in example shown below, is also used in determining whether the value of Quant should be increased, decreased, or left unchanged.

The following describes control of the quantization parameter, Quant, for an example where the virtual buffer size is 16 Kbits. In this example, "MaxBitsPerGroup" represents the pre-determined data rate in bits per group. Offset[ ] is a set of values that adjust the "target_activity_level" according to the fullness of the buffer model, which is represented by "Buffer_fullness", and which is compared to various threshold values represented by BufTh1, BufTh2, and so on:

```
// Set target number of bits per Group according to buffer fullness
if(Buffer_fullness < BufTh1)
{
```

-continued

```
        Target_activity_level = MaxBitsPerGroup + offset[0];
        MIN_QP = minQP[0];
        MAX_QP = maxQP[0];
}
else if(Buffer_fullness < BufTh2)
{
        Target_activity_level = MaxBitsPerGroup + offset[1];
        MIN_QP = minQP[1];
        MAX_QP = maxQP[1];
}
else if(Buffer_fullness < BufTh3)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[2]));
        MIN_QP = minQP[2];
        MAX_QP = maxQP[2];
}
else if(Buffer_fullness < BufTh4)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[3]
));
        MIN_QP = minQP[3];
        MAX_QP = maxQP[3];
}
else if(Buffer_fullness < BufTh5)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[4]
));
        MIN_QP = minQP[4];
        MAX_QP = maxQP[4];
}
else if(Buffer_fullness < BufTh6)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[5]));
        MIN_QP = minQP[5];
        MAX_QP = maxQP[5];
}
else if(Buffer_fullness < BufTh7)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[6]));
        MIN_QP = minQP[6];
        MAX_QP = maxQP[6];
}
else if(Buffer_fullness < BufTh8)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[7]));
        MIN_QP = minQP[7];
        MAX_QP = maxQP[7];
}
else if(Buffer_fullness < BufTh9)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[8]));
        MIN_QP = minQP[8];
        MAX_QP = maxQP[8];
}
else if(Buffer_fullness < BufTh10)
{
        Target_activity_level = max(0, (MaxBitsPerGroup + offset[9]));
        MIN_QP = minQP[9];
        MAX_QP = maxQP[9];
}
else if(Buffer_fullness < BufTh11)
{
        Target_activity_level = max(0, (MaxBitsPerGroup +
offset[10]));
        MIN_QP = minQP[10];
        MAX_QP = maxQP[10];
}
else if(Buffer_fullness < BufTh12)
{
        Target_activity_level = max(0, (MaxBitsPerGroup +
offset[11]));
        MIN_QP = minQP[11];
        MAX_QP = maxQP[12];
}
else if(Buffer_fullness < BufTh13)
{
        Target_activity_level = max(0, (MaxBitsPerGroup +
offset[12]));
        MIN_QP = minQP[12];
}
else if(Buffer_fullness < BufTh14)
{
        Target_activity_level = max(0, (MaxBitsPerGroup +
offset[13]));
        MIN_QP = minQP[13];
        MAX_QP = maxQP[13];
}
else
{
        Target_activity_level = max(0, (MaxBitsPerGroup +
offset[14]));
        MIN_QP = minQP[14];
        MAX_QP = maxQP[14];
}
```

The 14 values of threshold (BufTh1 through 14) of buffer fullness in units of bits may be set for a virtual buffer model size of 16 Kbits (16,384 bits) as {1792, 3584, 5376, 7168, 8960, 10752, 12544, 13440, 14336, 15232, 15456, 15680, 15960, 16240}. The 15 values of offsets (offset[0 to 14]) for Target_activity_level may be set as {20, 10, 0, −2, −4, −4, −8, −10, −10, −10, −10, −12, −12, −12, −12}.

At any range of buffer fullness, which is bounded by two consecutive thresholds, e.g. BufTh1<=Buffer_fullness<BufTh2, there is a range of Quant, specified by MIN_QP and MAX_QP, allowed for the rate controller **208** to use. This helps to regulate the variation of Quant to avoid over-quantization when the buffer level is low, as well as avoiding the use of too many less significant bits that may not help with visual quality when the buffer fullness is high. The pair of parameters, MIN_QP and MAX_QP, associated with each range of buffer fullness levels are selected respectively from an array of 15 values of minQP[0 to 14], with example default values of {0, 0, 1, 2, 2, 3, 4, 8, 8, 8, 13, 14, 15, 16, 17}, and an array of 15 values of maxQP[0 to 14] with example default values of {2, 2, 2, 3, 3, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17}, according to the buffer fullness level.

The value of Quant is adjusted according to the measured activity levels, the target activity level, the allowed Quant range specified by MIN_QP and MAX_QP, and the strength of a strong edge. When there is a strong edge, the activity level normally increases significantly if the value of Quant stays fixed. The rate control algorithm detects the presence of a strong edge by examining the activity level of the current Group and that of the preceding Group as well as the associated values of Quant. When a strong edge is detected, the rate control algorithm does not increase the value of Quant immediately after the presence of the strong edge, in order to avoid potential quantization noise that is more readily visible in smooth areas that may follow a strong edge. This factor may be observed for example in some cartoon content. The rate control may increase the value of Quant at the second group after a strong edge. One parameter that serves as a threshold in detecting strong edges is defined as EdgeFactor in the pseudo code below.

Some implementations avoid excessive fluctuation of Quant around a high quantization value, which could result in visible high frequency quantization noise in some images. These implementations regulate the increase of Quant so that Quant does not increase for two consecutive Groups of pixels when the value of Quant is already high, with certain exceptions. However, the decrease of Quant may be allowed as soon as the measured activity level is low. These adjustments are controlled by two parameters defined as QuantIn-

crLimit[0] and QuantIncrLimit[1] in the example below; their default values may be set to 11. In the following example, RcSizeGroup represents the activity level, BitsCodedCur represents the actual number of bits used to code the most recently coded Group, and RcTgtBitsGroup represents the Target_activity_level. RcTgtBitOffset[0] and RcTgtBitOffset[1] are offset values that adjust the range of the target activity level. EdgeFactor is a parameter that is used to detect a strong edge. The quantization step of the last Group is Quant, which is saved as QuantPrev before it is assigned the value for the current Group.

The operation of the Quant adjustment may be implemented as follows:

```
if ( RcSizeGroup < (RcTgtBitsGroup - RcTgtBitOffset[0])
  && BitsCodedCur < (RcTgtBitsGroup - RcTgtBitOffset[0]))
{       QuantPrev = Quant;
        Quant = MAX(MIN_QP, (Quant-1));
}
else if (BitsCodedCur > RcTgtBitsGroup + RcTgtBitOffset[1])
{       if ((QuantPrev == Quant && RcSizeGroup * 2 <
RcSizeGroupPrev * EdgeFactor) || (QuantPrev < Quant &&
RcSizeGroup < RcSizeGroupPrev * EdgeFactor &&
Quant < QuantIncrLimit[0])
|| (Quant < QuantIncrLimit[1] ) )
        {       QuantPrev = Quant;
                Quant = MIN(MAX_QP, (Quant+1));} }
else   QuantPrev = Quant;
```

When the buffer fullness approaches the maximum allowed level, the above Quant value determined by the activity level may be replaced by max_QP:

```
        if (Buffer_fullness >= BufTh_overflow_avoid)
            *Quant = max_QP;
```

Where BufTh_overflow_avoid is a programmable parameter.

FIG. **8** shows an example decoder **800**. The decoder **800** includes a rate buffer **802**, a variable length decoder (VLD) **804**, a predictor, mapper and inverse quantizer (PMIQ) **806**, and a rate controller **808**. The decoder **800** may be located in the same device or in a different device as the encoder, and may receive the bitstream input from any source, such as a memory or communication interface. For example, the decoder **800** may be located remotely from the encoder and receive the input bitstream via a network interface.

The rate buffer **802** may be a FIFO memory which temporarily stores compressed data bits after the encoder **800** receives them. The rate buffer **802** may be integrated with the rest of the video decoder or it may be located in another module, and it may be combined with another memory. The size of the rate buffer **802** may be at least as large as the virtual buffer used in the video encoder. For example, where the video encoder uses a 16 kbits virtual buffer, e.g., 2048 bytes, the rate buffer may be the same size, i.e., 2048 bytes or larger. Ready-accept flow control may be used between the rate buffer **802** and the VLD **804** to control that when the rate buffer **802** is empty the decoding operation is suspended until there is data available in the rate buffer **802**.

The fullness of the rate buffer **802**, at any given time, may not be the same as the fullness of the virtual buffer model. In part this is because the decoder virtual buffer model mimics the operation of the encoder virtual buffer model, and not the operation of the decoder, and the buffer model operates with the specified number of coded bits/pixel times the number of pixels in a Group being removed from the buffer model every time a Group is decoded, rather than the actual schedule at which bits arrive at the decoder. The transmission of compressed bits may be modeled as being exactly synchronized with the decompression function, while in actual operation the input of the rate buffer **802** may be read from memory more quickly or more slowly than exactly this rate. This is one reason that the rate control, above, operates on the buffer model and not on the rate buffer fullness.

The input to the VLD **804** is a compressed bit stream **812**. The compressed bit stream **812** nay include a series of Groups. The Groups may include a set of Units. Each Unit may have a Prefix and some number of samples, for example two, three or four samples. The VLD **804** operation is the inverse of the variable length coder (VLC) **206** function. Since the input to the VLD **804** is a stream of bits, e.g., a stream of VLC coded samples, part or all of the VLD operation may be performed sequentially. Some of the VLD functionality may be pipelined, however.

In one implementation, the VLD **804** uses a Unit size of 2, i.e., 2 samples per Unit. The choice of Unit size may be the same for both the encoder and decoder for any given image. The Unit size is generally an attribute of the encoded bit stream.

The VLD **804** decoding operation entails determining the actual sizes (e.g., number of significant bits) of the samples in the previous Unit of the same component as the one currently being coded, and creating a predicted Unit sample size from this information. This analysis may be pipelined. The VLD **804** may decode the Prefix of each unit, which may be unary coded. The decoded Prefix value is added to the predicted sample size value. The resulting sample size information indicates how many bits for each sample are contained in the Unit. The VLD **804** extracts from the incoming bit stream a number of bits equal to the prefix size plus the determined sample size times the number of samples per Unit. Once the VLD **804** extracts these bits, they are de-multiplexed and processed by subsequent decoding steps which may be pipelined.

Similar to the VLC, the number of bits spent for the current Group as well as the activity level of the current Group are calculated and passed to the rate controller **808** for rate control. The VLD **804** generates the values of RcSizeGroup and BitsCodedCur and passes these to the rate controller **808**.

Once the coded samples are extracted, they are converted to a suitable format for subsequent processing. For example, they may be converted to an 11 bit 2's complement signed format, with sign-extension of negative sample values. These constant-width sample values are demultiplexed into individual component streams of samples, and sent to the Predictor, Mapping and I-Quant (PMIQ) block **806**.

FIG. **9** shows example logic **900** for encoding. The logic **900** initializes the quantization step to zero (**902**) and receives a unit of pixel components (**904**). The logic **900** also performs quantization using the quantization step and encodes the quantized values (**906**). The logic **900** measures the fullness of the virtual buffer (**908**) and adjusts the quantization step based on the measured fullness (**910**). If the encoding is finished (**912**), flow may return to (**902**) or terminate altogether; otherwise flow may continue at (**904**).

FIG. **10** shows example logic **1000** for decoding. The logic **1000** initializes the quantization step to zero (**1002**). The logic **1000** decodes a coded unit and updates the virtual buffer (**1004**). The logic **1000** also dequantizes using the quantization step parameter (**1006**), and measures the full-

ness of the virtual buffer (**1008**). Further, the logic **1000** may adjust the quantization step based on the measured fullness (**1010**). The logic **1000** determines whether decoding of the frame is finished (**1012**), and if so, flow may return to (**1002**) or terminate. Otherwise, the flow may return to (**1004**).

Operation Description

The description above provides an example architecture that supports additional specific image processing operations. An introduction to some of these operations is provided next. Additional architectural implementations that support the image processing operations are also discussed further below.

FIG. **11** shows an example encoding and decoding system **1100**, based on the example of FIG. **1**. The system **1100** supports real time operation. Source data **112**, which may be uncompressed, enters the encoder **104**, for example in real time and raster scan order. The encoder **104** compresses incoming pixels to form a bitstream and temporarily stores portions of the bitstream in its rate buffer **210**. The output of the rate buffer **210** is the slice layer of a Display Stream Compression (DSC) bitstream **1106**. The DSC bitstream **1106** may be conveyed, e.g., in real time from the encoder **104** to the decoder **106**. In that regard, a wide variety of communication links **1104** may convey the DSC bitstream **1106** to the decoder **106**. Underlying the communication links **1104** may be a wide variety of transport layers, and the communication links **1104** may include local high speed busses, WiFi links, Ethernet links, satellite links, cellular (e.g., 3G or 4G/LTE) links, as examples.

The decoder **106** receives the DSC bitstream **1106** into its rate buffer **802**, which temporarily stores portions of the DSC bitstream **1106**. The decoder **802** decodes bits from the rate buffer **802** to obtain uncompressed pixels. The decoder **802** outputs the uncompressed pixels, e.g., in real time and in raster scan order, for the display **110**. The image output from the decoding process may have the same format as the image input to the encoding process.

The DSC bitstream may include of a sequence of frames coded using a picture layer syntax. The picture layer syntax may include a PPS (picture parameter set) and a slice syntax. The PPS contains parameters that the decoder **106** uses for correct decoding of the slice layer. FIG. **12** shows an example of a PPS **1200**.

The picture layer may operate in units of entire pictures. A picture may be, as examples, a frame in the case of a progressive format video, or a field in the case of an interlaced format video. Each picture may include an integer number of contiguous, non-overlapping, identically-sized, rectangular slices. In the encoder **104**, slice coding is specified via a slice layer. In the decoder **106**, each slice may be decoded independently without reference to other slices. There may be one slice per line or multiple slices per line. In the case of multiple slices per line, bits from the slices covering one line are multiplexed in the DSC bitstream **1106** via a slice multiplexing process described below. Each slice may include a set of groups, and each group may be a set of three consecutive pixels in raster scan order. Further, the encoder **104** may encode each group with multiple (e.g., three) entropy codes, one for each component, and each of which may be a specific type of variable length code (VLC). Furthermore, some groups may include one or more additional bits which signal specific decoding operations.

FIG. **13** shows another example of an encoder **1300**. The DSC encoding process generates bitstreams that may precisely conform to the independently specified bpp (bits per pixel) rate. The bpp rate may be specified in terms of bits per pixel time, which may be algorithmically specified, as the unit of a pixel time is the same at both the input and output of the encoder **1300**. The number of bits that code each pixel, or group of pixels, may vary considerably. In the encoder **1300**, the rate buffer **1302** facilitates converting the variable number of bits used to code each group into, e.g., a constant bpp rate. To that end, the encoding process includes the rate controller **1304**.

The encoder **1300** may include color space conversion logic **1306**, e.g., RGB input to reversible YCoCg conversion logic. An input buffer **1308** stores the converted input. Prediction, quantization, and reconstruction (PQR) logic **1310** implements prediction of sample values and generation of residual values. The prediction, quantization, and reconstruction (PQR) logic **1310** may include multiple (e.g., three) predictors: modified median adaptive prediction (MMAP), mid-point prediction (MPP), and block prediction (BP). The PQR logic **1310** also implements quantization of residual values and reconstruction of sample values. An indexed color history (ICH) **1312** is also present, as is VLC coding logic **1314** that may implement entropy coding using delta size unit variable-length coding (DSU-VLC). The input buffer **1308** provide samples to the flatness determination logic **1318**. Note also that substream multiplexing logic **1320** is present to prepare a multiplexed output stream to the rate buffer **1302**.

FIG. **14** shows another example of a decoder **1400** configured to decode image data that the encoder **1300** has encoded, and produce image output **1418**. The decoder **1400** may implement the inverse of the operations that were performed by the encoder **1300**. The decoder **1400** may include a rate buffer **1402**, substream demultiplexer **1420**, and VLC entropy decoding logic **1404** for delta sized unit variable length coding (DSU-VLC). The decoder **1400** also includes PQR logic **1406** that may implement multiple (e.g., three) predictors: modified median adaptive prediction (MMAP), mid-point prediction (MPP), and block prediction (BP). The PQR logic **1406** also performs inverse quantization of residual values and reconstruction of sample values. An ICH **1408**, rate control logic **1410**, and color space conversion logic **1412** is also present. Flatness indications may be signaled in the bitstream from the encoder, and provided to the rate control logic **1410**.

The encoding process may produce display stream coded bitstreams that conform to an HRD (hypothetical reference decoder) constraint. The HRD may be idealized model of a decoder that includes a model of a rate buffer, which should neither overflow nor underflow.

The DSC bitstream and decoding process facilitate decoding 3 pixels per clock cycle in practical hardware implementations. In other implementations, the decoding process may process 1, 3, or other numbers of pixels per clock. Additional throughput in terms of pixels per clock may be increased via encoding and decoding multiple slices in parallel, which is facilitated by utilizing multiple slices per line in the DSC bitstream.

Color Space Conversion Logic **1306**, **1412**

RGB video input to the encoding process may be converted to YCoCg for subsequent processing. The reversible form of YCoCg may be used, and as such the number of bits per each of the two chroma components is one greater in YCoCg than it is in RGB. In the case of YCbCr input, no color space conversion need be performed. The inverse color space conversion is performed in the decoding process.

PQR Logic **1319**, **1406**

Each group of pixels is coded using either predictive coding (P-mode) or indexed color history coding (ICH-mode). For P-mode there are three predictors: modified

median-adaptive prediction (MMAP), block prediction (BP), and midpoint prediction (MPP). The encoder and decoder may select MMAP, BP, or MPP automatically, using the same algorithm in each, without signaling the selection in the DSC bitstream.

In the encoder **1300**, each sample is predicted using the selected predictor. The original sample value is compared to the predicted value, and the difference is quantized. Each quantized error is then entropy-coded if P-mode is selected. The encoder **1300** also performs a reconstruction step wherein the inverse-quantized error is added to the prediction so that the encoder and decoder may use the same reference samples.

In decoder **1400**, the samples are predicted using a selected predictor. The residual value, which is obtained from decoding the DSC bitstream, is inverse quantized and the result added to the prediction, forming the reconstructed sample value.

The median-adaptive predictor (MAP) may be the prediction method that is used in JPEG-LS. However, a modification is made to allow the decoder **1400** to process three pixels in a group in parallel and to improve coding. The modified median-adaptive predictor (MMAP) facilitates hardware implementations for decoders running at 3 pixels/clock. The MMAP predicts a current sample value as a function of reconstructed previously coded samples to the left and above the current sample. The encoder **1300** and decoder **1400** may use identical sets of reconstructed samples for this purpose, and hence the MMAP produces the same results in both the encoder **1300** and the decoder **1400**. MMAP may be the default predictor, and is effective at predicting sample values in most conditions.

The MPP predicts a current sample from a value that is approximately at the mid-point of the valid range for the sample. The MPP has the benefit of bounding the maximum size of the residual. MPP may be selected in place of MMAP when the number of bits required to code the samples in of one component of a group would be greater than or equal to the bit depth for that component minus the quantization shift.

The BP predicts a current sample from a reconstructed previously coded sample to the left of the current sample in the same scan line. The offset from the current sample to the predictor position is a BP vector. The BP vector and the decision of whether or not to use BP are determined automatically by the BP function, which is the same in both the encoder and decoder.

Block Prediction

Block prediction may predict the current sample where the predictor is a sample to the left of the current sample, in the same line. The relative position of the reference sample may be between (−3) and (−10), inclusive. Using additional pixel locations may improve quality. The relative position is a vector within the same line of samples; this is referred to as the block prediction vector.

The search to find the best vector may be performed on the previous line of samples, rather than the line that is currently being coded. In one implementation, the block search compares a set of 9 consecutive samples with reference samples using various potential vectors with values ranging from −3 to −10. The current samples and the reference samples being compared are in the same scan line, e.g., the line above the line of the sample to be coded. For each vector considered, a SAD (sum of absolute differences) is calculated over 9 samples in each of the current and reference set. The vector with the lowest SAD value is selected. In cases of ties, the vector closest to 0 is selected.

The 9-pixel SAD of the vector −1 is also used in order to determine whether BP or MMAP should be used. More details of predictor selection are given below.

A vector, once selected, applies to each group of 3 samples. Therefore the block search is performed every 3 samples.

A vector means that the predictor for pixel X is the pixel that is to the left of pixel X in same line, the distance to the left in pixel units being equal to the vector value.

FIG. **15** illustrates example sample sets **1500** for block search, showing several reference samples **1502** and vectors **1504**, **1506**. An example of the current sample 'x' **1506** and the current SAD calculation samples **1508** are also shown.

Indexed Color History (ICH) Logic **1312**, **1408**

FIG. **16** illustrates an example of indexed color history **1600**.

In many types of content, such as computer-generated text and graphics, similar pixel values tend to appear in reasonably close proximity while not necessarily being adjacent to one another. Because of this, it can be helpful to keep track of a number of recently-used pixel values in the Indexed Color History (ICH). When the encoder **1300** selects ICH-mode for a particular group, it sends index values corresponding to the selected pixel values within the ICH. These index values are used directly in the output pixel stream.

The ICH logic includes a storage unit that maintains a set of recently used color values that were coded using another coding method such as predictive coding. The encoder **1300** and decoder **1400** may maintain identical states of the ICH. The ICH may have 32 entries, with an index value pointing to each entry. For groups that are ICH coded, each pixel may be coded with a 5-bit ICH index, which points to one of the entries. As each group of pixels is encoded in the encoder or decoded in the decoder in P-mode, the values of all the pixels in the group are entered into the ICH. The ICH may be managed as a shift register where the most-recently used (MRU) values are at the top and the least-recently used (LRU) values are at the bottom. New entries are added at the top and all other entries are shifted down, with the bottom entries falling out of the ICH. When a group is coded in ICH-mode, the three indices used to code those pixels reference entries in the ICH. When an ICH entry is referenced, it is moved to the top of the ICH and the other values above the prior location of the entry are shifted down by 1. This operation is performed in parallel for all 3 entries of each ICH coded group, and the most recent, e.g., the rightmost pixel value of the group becomes the MRU. The result is that the most recently used (MRU) value is at the top of the history and the least recently used (LRU) value is at the bottom of the history. Whenever a P-mode group of three pixels is added at top of the history, the three LRU values are removed.

For the first line each slice, all 32 ICH entries are treated as part of the shift register. For lines after the first line of a slice, the last 7 index values are defined to point to reconstructed pixels in the line above the current line, rather than entries in the ICH. This is useful for efficient coding of pixel values that are not in the history shift register, and it improves coding with some content.

ICH mode may be selected on a per-group basis by the encoder **1300**. The encoder **1300** signals the use of ICH mode for a group using an escape code in the luma substream DSU-VLC. For each group coded in ICH mode, each pixel in the group is coded using a fixed-length 5 bit code, where the index values point into the history. The decoder **1400** decodes each ICH-coded group by determining the use of ICH mode via the bitstream syntax and decoding each

pixel in the group by reading the values pointed to by the ICH indices that constitute the coded values of the pixels. Both the encoder **1300** and decoder **1400** update the ICH state identically every group by inserting P-mode pixels into the ICH and by re-ordering the ICH entries in response to ICH mode groups.

Entropy Coding Logic **1314**, **1404**

The display stream coding defines syntax at multiple layers. The lowest layer is called the substream layer. There may be three substreams in each slice, one for each component. The three substreams may be multiplexed together by a substream multiplexing (SSM) process to form a coded slice. If there is more than one slice per line, the coded slices may be multiplexed by the slice multiplex process; and if there is only one slice per line, the slice multiplex process is not used. The resulting bits of all slices are concatenated to form a coded picture. Each coded picture is optionally preceded by a picture parameter set (PPS).

Substream Layer

The display stream encoding may use an entropy coding technique referred to above as DSU-VLC for coding residuals associated with predictive coding. ICH coding of pixels uses a fixed-length code for each pixel. Specialized values are used to signal the use of ICH mode, and other codes signal quantization adjustments associated with flat regions of pixels.

TABLE 1

Examples of sizes for different residual values

Residual values	Size in bits	Representation
−3	3	101b
−2	2	10b
−1	1	1b
0	0	<none>
1	2	01b
2	3	010b
3	3	011b

The pixels in each slice may be organized into groups of three consecutive pixels each. A group is a logical construction employed by the encoding and decoding processes, but need not be directly represented in the bitstream. DSU-VLC organizes samples into units. A unit is the coded set of residuals of three consecutive samples of one component. Each unit has two parts: a prefix and a residual. The size of each residual is predicted based on the size of the three previous residuals of the same component type and any change in QP that may have occurred. The prefix may be a unary code that indicates the non-negative difference between the size of the largest residual in the unit and the predicted size. If the difference is negative, the value coded by the prefix is zero. The residual portion of each unit contains 3 values, one for each sample in the unit. The residual values are coded in 2's complement. The number of bits allocated to residuals can vary from unit to unit; however, all 3 residuals in one unit may be allocated the same number of bits.

In addition, the prefix for luma units also indicates whether or not ICH mode is used for each group. A transition from P-mode to ICH-mode may be indicated by an escape code, e.g., a prefix value that indicates a size that is one greater than the maximum possible residual size for luma. The maximum possible residual size for luma depends on the QP value that applies to luma in the group. An ICH-mode group immediately following another ICH mode group may be indicated by a luma prefix code consisting of a single "1" bit. A P-mode group immediately following an ICH-mode group may be indicated by a modified unary code.

For an ICH-mode group, the residual portion may be 5 bits for each component, where each 5 bit code is an ICH index which codes a complete pixel, and the chroma components do not utilize a prefix. For subsequent ICH-mode groups following an initial ICH-mode group, each group may use 16 bits for every group, e.g., a 1 bit prefix and (3) 5 bit ICH codes.

The luma substream may also contain some conditional fixed-length codes in the syntax for the purpose of the encoder conveying information about a transition from a busy area to a smooth area. This "flatness indication" is discussed in more detail below.

Substream Multiplexing

The three component-wise substreams may be multiplexed together using a fixed-length substream multiplexing scheme with no headers. One technique for doing so is described in the U.S. Patent Publication Number 2011-0305282 A1, which is incorporated by reference. FIG. **17** shows an example of the results of substream multiplexing **1700**, including various multiplexed words and components **1702**. Each mux word may have an identical size, e.g., 48 bits for 8 or 10 bits per component (bpc), or 64 bits for 12 bpc. The order of the mux words **1702** is derived from the order in which parallel substream decoders use the data in order to decode in real time.

FIG. **18** shows an example of substream demultiplexing logic **1800**. The logic **1800** includes a memory such as a rate buffer **1802**, a demultiplexer **1804**, and funnel shifters with VLD **1806**, **1808**, and **1810**. The combination of the funnel shifter and VLD is referred to as a substream processor (SSP). At each group time, any combination of the SSP's may request a mux word or none at all. If a request is received from an SSP, the demultiplexer **1804** sends a mux word to that SSP. If multiple requests are received in the same group time, the demultiplexer **1804** sends a mux word to each SSP that made a request.

At the end of the slice, the SSP's may request mux words beyond the end of the substream layer data. Therefore, the encoder **1300** may insert padding mux words as needed at the end of the slice.

FIG. **19** shows an example of the substream multiplexing logic **1900**, including VLC and funnel shifters **1902**, **1904**, **1906**, balance memories (e.g., FIFOs) **1908**, **1910**, **1912**, a multiplexer **1914**, rate buffer **1916**, and demultiplexer model **1918**. The demultiplexer model **1918** helps the encoder **1300** to order the mux words correctly. The balance FIFO's **1908**, **1910**, **1912** may store many groups worth of data in order to provide the mux words at the appropriate time.

Rate Control

The encoder **1300** and decoder **1400** may use identical rate control (RC) algorithms, configured identically. The decisions made by the RC algorithm to adjust QP in the encoder are mimicked in the decoder **1400**, such that the decoder **1400** has the same QP value as the encoder **1300** at every pixel, without any bits being spent communicating the QP value, except for the flatness indication. RC decisions are made in the encoder **1300** and decoder **1400** based on information previously transmitted and received. RC can change the QP value every group.

Rate Control Goals

The RC provides the encoder **1300** and decoder **1400** with quantization parameters (QP) to use for each group. Since the RC function is the same on both the encoder side and the decoder side, the base QP value is known to both encoder **1300** and decoder **1400**, and it does not need to be trans-

mitted in the bitstream. However, the base QP value or adjustments to the QP value may be sent in the bitstream for flatness indication, described below.

The RC attempts to ensure hypothetical reference decoder (HRD) conformance. There is a model of an idealized rate buffer (FIFO) that converts a varying number of bits to code each group into a specified constant bit rate. The RC is designed to ensure that this FIFO will not overflow or underflow assuming that bits are removed at an assumed constant bit rate.

The RC optimizes picture quality in its QP decisions. It is desirable to use a lower QP on relatively flat areas and a higher QP on busy areas due to perceptual masking. In addition, it is desirable to maintain a constant quality for all pixels; for example, the first line of a slice has limited prediction, and may therefore use an additional bit allocation.

HRD Buffer Model

A hypothetical reference decoder (HRD) model describes the behavior of an idealized rate buffer in a decoding system. An encoder rate buffer model may be mirrored on the decoder side. The encoder model tries to ensure that there are no overflows or underflows. Since the DSC may be constant bit rate (CBR), the HRD model fullness is equal to buffer size—encoder buffer fullness; therefore, the decoder buffer model does not overflow or underflow. The DSC encoder rate buffer model may define a schedule for bits entering and leaving the rate buffer.

During the initial delay, e.g., initial transmission delay, the encoder generates bits into its rate buffer every group, but no bits are removed. During this period, the encoder model fullness increases according to the number of bits that are generated. The delay period may be specified in terms of group times or pixel times, as examples.

As long as there are more pixels in the slice to be encoded, the encoder generates bits according to the content. Bits are removed at the constant rate that is specified. To prevent the buffer fullness from dropping below 0, the prediction mode may be overridden to use MPP, which enforces a minimum data rate. Once the last group of a slice has been encoded, no more bits are added to the rate buffer. Bits continue to leave the rate buffer at the constant rate until the buffer becomes empty, after which the encoder sends zero bits to ensure that the compressed slice size in bits is equal to bpp*number of pixels in slice, in CBR operation.

The decoder initial delay is specified as the complement of the encoder initial delay; e.g., the HRD delay minus encoder initial delay. The decoder rate buffer fullness then tracks as the complement of the encoder buffer fullness.

CBR vs. VBR

Under conditions when the encoder rate buffer would otherwise underflow, there is a design choice of whether the encoder inserts bits to prevent underflow, or it uses VBR. To prevent underflow, the RC determines whether underflow is possible after the next coded group, and when this condition occurs it forces MPP mode which enforces a minimum bit rate. The decoder does not require any special logic to handle stuffing, as it decodes the extra bits just as it would any other group.

It is possible to support variable bit rate (VBR). With VBR, the encoder **1300** stops sending bits under certain conditions when it would otherwise underflow and has no bits to send (Off). The encoder **1300** then starts sending bits again at some identified event (On). To make on-off VBR compatible with a general HRD that does not depend on the real time behavior of the transport, the off and on events may be specified.

With VBR, the encoder stops sending bits when it would otherwise underflow and has no bits to send. The encoder's RC process operates once per group. At each group, it adds to the buffer model the number of bits that code the group, and normally it subtracts from the buffer model the nominal number of bits per group, which is 3*bpp, adjusted as necessary to form an integer number of bits. With VBR, if this subtraction of bits/group from the buffer model fullness would result in a negative value of fullness, the RC subtracts the normal number of bits and then clamps the buffer fullness to zero, i.e. the model fullness is never allowed to be negative. In a real system with a real transport and real decoder, when the encoder has no bits to send, i.e. its real rate buffer is empty, the transport does not send any bits and the decoder does not receive any bits. The decoder's real rate buffer may be full, but it does not overflow. When the encoder does have bits to send, transport is expected to transmit them at the normal rate and the decoder receives them at that rate. The decoder's real buffer does not overflow nor underflow, and the decoder does not have to do anything special to handle VBR. The transport should understand when there is and is not valid data available to send and receive.

Slices

The number of bits that code a picture may be equal to the number of pixels of that picture times the specified bpp rate. Further, any subset of slices of a picture may be updated in place in a compressed frame buffer by over-writing the previous version of each of the corresponding slices. One consequence is that a complete picture can be transmitted as a series of consecutive slices comprising the entire picture, and that an entire picture transmitted as a series of consecutive slices meets the same requirement as for slices, e.g., the number of bits equals the number of pixels times the bpp rate, and also the entire picture comprising slices should conform to an appropriate HRD model to ensure correct real time buffer behavior with this mode of operation. One consequence is that the delay from the start of transmission to the start of decoding and the delay from the end of transmission to the end of decoding are the same as one another and the same for each slice.

The algorithm uses a rate buffer model, which may be referred to as a rate buffer. The algorithm allows the encoder's rate buffer to have up to a specified fullness, e.g., a maximum number of bits, at the end of each slice. If at the end of coding a slice the encoder's buffer has fewer bits than this maximum number, it may pad the remaining bits at the end with 0s, for example, to produce exactly the required number of bits. This final number of bits occupies a specified number of pixel times to transmit at the specified bpp rate. This number of pixel times is the delay from the end of encoding to the end of transmission, which may be called the final transmission delay. The total rate buffer delay, in units of pixel times, in the combination of an idealized encoder and decoder is equal to the rate buffer size divided by the bpp rate. The initial transmission delay, from the start of encoding a slice until the start of transmission of that slice, is the same as the final transmission delay. The initial decoding delay, e.g., the delay in the HRD timing model from the start of reception of a slice to the start of decoding of the slice is set equal to the total end-end rate buffer delay minus the initial transmission delay. This permits correct operation per the description above.

FIG. **20** shows an example of slice timing and delays **2000**. FIG. **20** shows slice input video timing **2002**, slice transmission timing **2004**, and slice decoding timing **2006**. The algorithm may have a fixed parameter value for the

maximum number of bits that can be in the encoder buffer at the end of a slice, typically ~4 kbits. The resulting ending transmission delay is a function of the bpp rate; it is set to ceiling (4096/bpp_rate). At 8 bpp, this delay is 170 group times, and at 12 bpp it is 114 group times. The initial delay may be set to this value.

The end-end HRD delay is equal to the HRD buffer size divided by the bpp rate. For example, if the HRD buffer size is 19,836 bits and the rate is 12 bpp, the end-end HRD delay is floor (19,836/36)=551 group times. This is actually an upper bound, and the HRD delay could be set to a lower value, however if a lower value were used then the algorithm would not be able to take full advantage of the available buffer size for purposes of RC.

The initial decoding delay, which applies directly to the HRD and indirectly to real decoders, should be set to the HRD delay—initial transmission delay. In the example here, where the initial transmission delay is set to 114 group times as above, the initial decoder delay is 551−114=437 group times. This is a delay that applies to the HRD, i.e. an idealized hypothetical decoder. A real decoder is of course free to have additional delay.

The algorithm's rate buffer size, which is also the HRD buffer size, can be selected by an encoder as long as it does not exceed the capabilities of compatible decoders. The optimum rate buffer size is a function of several factors including the bpp rate and the width of slices.

Note that the initial transmission delay is typically a function of bpp rate. The HRD rate buffer size may be set by the encoder as long as it does not exceed the capabilities of decoders. It is practical to design real systems with adjustable bit rate and constant end-end delay, from video into the encoder to video out of the decoder, and with constant delay from compressed data into the decoder to video put of the decoder. An encoder may set the initial transmission delay and the initial decoder delay to selected values to facilitate seamless changes of bit rate with constant delay.

Options for Slices

The encoder **1300** and decoder **1400** support a wide variety of slice widths and heights. One configuration is slice width=¼ picture width and slice height=32 lines. Another possible configuration is slice width=picture width and slice height=8 lines. The slice dimensions can be specified up to the picture width by the picture height. To minimize extra data that may need to be sent, equal-sized slices may be used throughout the picture.

Taller slices may lead to better compression. Extra bits are allocated to the first line of each slice to maximize quality and to prevent artifacts at the boundaries between slices. The number of extra bits allocated per group on the first line is set via a parameter in the PPS. The numbers of bits available to all lines after the first line each slice may be reduced in order that the total number of bits per slice is the number of pixels times the bpp rate. The more lines there are after the first line in each slice, the less reduction in bit allocation is required. Therefore a slice height of 32 lines typically gives better performance than a slice height of 8. There is no cost associated with slice height—there is no additional buffering nor any other additional resources. The encoder **1300** and decoder **1400** support a slice size equal to the entire picture size.

Slices narrower than the full screen width may be desirable for various practical purposes. Narrower slices provide the ability to update, via partial update, a narrower slice, or to facilitate parallel processing at low cost. In practice, multiple slices per line can use one line buffer the size of the picture width. With multiple slices per line, and slices that are taller than one line, the rate buffers for the different slices may be independent. For example, with four slices per line, a practical implementation would use four rate buffers. The sizes of each rate buffer can be specified to be smaller for the case of 4 slices/line than they would normally be specified for the case of one slice/line, as the optimum rate buffer size is a function of the slice width, although not exactly proportional. Hence there is a small increase in the total amount of rate buffer space when there are multiple slices per line, while there is no increase in the total amount of line buffer space.

Slice Multiplexing

In systems configured to use more than one slice per scan line, the compressed data may be multiplexed according to a specific pattern in order to minimize cost in both encoders and decoders. The recommended pattern is as follows. For an integer number S of slices per line, each slice has P pixels per line, and the picture is W pixels wide. Preferably P is equal for all slices, equal to W/S, which is preferably an integer. The multiplexed bit stream contains a number of bits=P*bpp rate for the first slice of the first row of slices, then P*bpp rate for the 2nd slice of the first row, and so on for all slices of the first row.

One iteration of this pattern has W*bpp rate bits, which may be the same number of bits as would have been used if there were one slice per line. If P*bpp rate is not an integer, an adjustment can be made to result in an integer number of bits per slice. For example, the number of bits included for one line of one slice may be the integer truncated value of P*bpp plus the accumulated residual amount from previous truncations. Then this pattern repeats as many times as needed to transmit all the bits of all slices in the first row of slices. An application specification, for example a transport specification that is designed to carry DSC compressed image data, may carry data from different slices in separate packets. In that case, the last bits from one slice may be in a separate packet from those of other slices, including the first bits of the vertically adjacent slice immediately below the first one. Alternatively an application specification may choose to package the last bits of one slice with the first bits of another slice, for example a horizontally adjacent neighboring slice or a vertically adjacent neighboring slice. The overall pattern may repeat for the entire image. It is not necessary to include markers or other indications in the bit stream indicating which bits are for which slice. Instead, the transport layer may provide such indicators.

Additional information on slice multiplexing follows.

Slice multiplexing may occur when VBR is disabled, e.g., stuffing is enabled. When stuffing is disabled, the number of bits coding each slice may vary, e.g., the DSC operation is VBR. Pictures include some number of slices. Slices may be identically-sized when possible, e.g., when the ratio of picture width to slice width is an integer. In case this ratio is not an integer, the widths of the columns of slices may be set to integer values that differ by no more than 1, and whose sum is the picture width. Slice multiplexing is possible also when VBR is enabled as well. The memories used and multiplexing pattern will depend on characteristics of the link, including for example, the overhead required to enter or leave a low-power state.

With VBR disabled (stuffing enabled) slices of the same width are coded using the same number of compressed bits. When the slice width is equal to the picture width, the slice layer data is sent sequentially (slice 0, slice 1, . . . , slice N−1, where N is the number of slices). When the slice width is shorter than the picture width, the slice data for all slices on the same line may be multiplexed into fixed-length chunks.

The length of each chunk may be equal to floor (bits_per_pixel*slice_width). The floor( ) function is used since bits_per_pixel may be fractional. For example, in a case where the picture is split into two equal-sized slices on each line, the multiplexed bitstream would contain:

Slice 0 chunk/Slice 1 chunk/Slice 0 chunk/Slice 1 chunk . . . .

The final chunks of each slice may be padded with zero bits if needed due to the ceil( ) function.

With VBR enabled, the number of bits of coding each slice may differ from P*bpp rate. For example, the number of bits may be less than this value. The number of bits per chunk may differ from floor (bits_per_pixel*slice_width), for example the number of bits may be less than this value. Slices may be multiplexed using chunks of unequal numbers of bits. The numbers of bits per chunk may be indicated for example by packet length information or marker codes in a transport layer.

The display stream coding may be specified in terms of components that are labeled Y, Co, and Cg. If the convert_rgb flag is equal to 0 in the current PPS, the encoder may accept YCbCr input. The Cb component may be mapped to the Co component label. The Cr component may be mapped to the Cg component label. In this case, the bit depth of the Cb/Co and Cr/Cg components may be equal to the Y component, whose bit depth is specified using the bits_per_component field in the current PPS. If the convert_rgb flag is equal to 1 in the current PPS, the encoder may perform color-space conversion from RGB to YCoCg. The color space conversion may be:

$$cscCo=R-B$$

$$t=B+(cscCo>>1)$$

$$cscCg=G-t$$

$$y=t+(cscCg>>1)$$

The cscCo and cscCg values have one additional bit of dynamic range compared with Y. The final Co and Cg values may be centered around the midpoint:

$$Co=cscCo+(1<<bits\_per\_component)$$

$$Cg=cscCg+(1<<bits\_per\_component)$$

Note that here, the bits_per_component variable may represent the number of bits of each of the R, G, and B components, which is one less than the number of bits per component for the Co and Cg components. If a slice extends beyond the right edge of a picture, the right-most pixel in each line of the picture may be repeated to pad the slice to the correct horizontal size. If a slice extends beyond the bottom edge of a picture, the bottom-most pixel in each pixel column of the picture may be repeated to pad the slice to the correct vertical size.

Line Storage

The display stream compression may include buffer memory to hold the previous line's reconstructed pixel values for MMAP prediction and ICH. In some cases, a decoder line buffer may have sufficient storage to contain the full-range reconstructed samples. However, some decoders may choose to use a smaller bit depth to lower the implementation cost.

If a smaller bit depth is used, the decoder may communicate this to the encoder. The encoder may set the linebuf_width according to what the decoder implementation supports. The following method for bit-reducing samples may be used:

$$shiftAmount=MAX(0,maxBpc-linebuf\_width);$$

$$round=(shiftAmount>0)?(1<<(shiftAmount-1)):0;$$

$$storedSample=(sample+round)>>shiftAmount;$$

$$readSample=storedSample<<shiftAmount;$$

where maxBpc is the bit depth of the current component, storedSample is the sample value that is written to the line buffer, and readSample is the value that is read back.

Prediction Types

There are three prediction types that may be supported in P-mode: MMAP, BP, and MPP.

Modified Median-Adaptive Prediction (MMAP)

The modified median-adaptive predictor is specified in the table below.

TABLE 2

Pixels surrounding current group			
c	b	d	e
a	P0	P1	P2

Table 2 shows the labeling convention for the pixels surrounding the three pixels in the group being predicted (P0, P1, and P2). Pixels 'c', 'b', 'd', and 'e' are from the previous line, and pixel 'a' is the reconstructed pixel immediately to the left.

A QP-adaptive filter may be applied to reference pixels from the previous line before they are used in the MMAP formulas below. A horizontal low-pass filter [0.25 0.5 0.25] may be applied to the previous line to get filtered pixels filtC, filtB, filtD, and filtE. For example,

$$filtB=(c+2*b+d+2)>>2;$$

The filtered pixels may be blended with the original pixels to get the values that are used in MMAP (blendC, blendB, blendD, blendE). The following method is used for the blending:

$$diffC=CLAMP(filtC-c,-QuantDivisor[qlevel]/2,\ QuantDivisor[qlevel]/2);$$

$$blendC=c+diffC;\ diffB=CLAMP(filtB-b,-QuantDivisor[qlevel]/2,\ QuantDivisor[qlevel]/2);$$

$$blendB=b+diffB;$$

$$diffD=CLAMP(filtD-d,-QuantDivisor[qlevel]/2,\ QuantDivisor[qlevel]/2);$$

$$blendD=d+diffD;$$

$$diffE=CLAMP(filtE-e,-QuantDivisor[qlevel]/2,\ QuantDivisor[qlevel]/2);$$

$$blendE=e+diffE;$$

The predicted value for each is given below:

$$P0=CLAMP(a+blendB-blendC,MIN(a,blendB),MAX(a,blendB));$$

$$P1=CLAMP(a+blendD-blendC+R0,MIN(a,blendB,blendD),MAX(a,blendB,blendD));$$

$$P2=CLAMP(a+blendE-blendC+R0+R1,MIN(a,blendB,blendD,blendE),MAX(a,blendB,blendD,blendE));$$

where R0 and R1 are the inverse quantized residuals for the first and second samples in the group.

In the case of the first line of a slice, the previous line's pixels are not available. So the prediction for each pixel becomes:

```
P0=a;

P1=CLAMP(a+R0,0,(1<<maxBpc)−1);

P2=CLAMP(a+R0+R1,0,(1<<maxBpc)−1);
```

where maxBpc is the bit depth for the component that is being predicted.

Block Prediction (BP)

The BP predictor is a pixel value taken from a pixel some number of pixels to the left of the current pixel. The "block prediction vector" (bpVector) is a negative value that represents the number of pixels to the left to use for the prediction. In one implementation, the block prediction vector is always between −3 and −10 inclusive, which means that it uses samples outside of the current group.

The BP predictor is used to predict all three components from the pixel referred to by the block prediction vector:

```
P[hPos]=recon[hPos+bpVector];
```

So the predicted values for the 3x1 group correspond with the reconstructed pixels values for the 3x1 set of pixels that is pointed to by the block prediction vector.

Midpoint Prediction

The midpoint predictor is a value at or near the midpoint of the range, and depends on the value of the reconstructed pixel immediately to the left of the current pixel (pixel "a" in Table 2).

```
midpointPred=(1<<(maxBpc−1))+(a&((1<<qLevel)−
    1));
```

where maxBpc is the bit depth for the component being predicted, and qLevel is the quantization level that applies to the current component.

Predictor Selection

Block prediction is supported by the encoder **1300**. The encoder **1300** may choose to disable block prediction in the stream (e.g., because the attached decoder does not support block prediction or because the picture would not benefit from block prediction) by setting block_pred_enable in the PPS equal to 0. In this case, MMAP is selected over block prediction, and the algorithms in this section are not used.

The decision to use either BP or MMAP may be made on a group basis using information from the previous line. This means that the decision can be made up to a line time in advance of processing the current group if it helps the implementation. The group referred to in this section starts at a horizontal location of hPos pixels from the leftmost pixel column in the slice.

FIG. **21** shows an example **2100** of 3×1 partial SADs that form 9×1 SAD. First, a search may be performed to find the best block prediction vector. The reference pixels for the SAD may be the set of 9 pixels in the previous line starting at a horizontal location of hPos −6. The SAD is computed between the reference pixels and 9 different block prediction candidateVector's (−1, −3, −4, −5, −6, −7, −8, −9, and −10) pointing to the previous line's pixels. The 9-pixel SAD is computed as a sum of 3 3-pixel SAD's (see FIG. **21**). First, each absolute difference may be truncated and clipped before being summed in the 3-pixel SAD according to:

```
modifedAbsDiff=MIN(absDiff>>(maxBpc−7),0x3F);
```

where maxBpc is the bit depth for the current component.

The resulting 6-bit modifiedAbsDiff values are summed over each set of three adjacent samples and over the 3 components, resulting in a 10 bit value that represents the 3x1 partial SAD for one component; this 10-bit value is clamped to 9-bits (e.g., values greater than 511 are clamped to 511). Three 9-bit 3-pixel partial SAD's are summed to get the final 9-pixel SAD, which is an 11-bit number. The 3 LSB's of each 9x1 SAD are truncated before comparison:

```
bpSad[candidateVector]=MIN(511,sad3x1_0[candi-
    dateVector]+sad3x1_1[candidateVector]+
    sad3x1_2[candidateVector]);
```

The 9 9-pixel SAD's are compared to one another, and the lowest SAD may be selected, with ties broken by selecting the smallest magnitude block prediction vector. If the lowest SAD block prediction vector is −1, the bpCount counter is reset to zero and MMAP is selected for this group. If the lowest SAD block prediction vector is not −1, the candidate BP vector becomes the vector with the lowest SAD, and the bpCount counter is incremented unless hPos<9.

BP may be selected if the following conditions are all true:

The bpCount value is greater than or equal to 3.

lastEdgeCount is less than 9. The lastEdgeCount value represents the number of pixels that have gone by since an "edge" occurred. An "edge" occurs when ABS(current sample−left sample)>32<<(bits_per_component−8) for any component.

Selecting Between BP/MMAP and MPP

The encoder may decide whether to use BP/MMAP based on the size of the quantized residuals that would be generated if BP/MMAP were selected. For example, the encoder may determine the maximum residual size for BP/MMAP for each of the three components. If the maximum residual size for any component is greater than or equal to a threshold such as maxBpc−qLevel for that component, then MPP may be selected for that component.

In addition, the encoder may select MPP in order to enforce a minimum data rate to prevent underflow.

Quantization

The predicted value of each sample of the pixel is subtracted from the corresponding input samples to form the residual sample values E, one for each component of the pixel.

$E=x-Px$, where x is input, Px is predicted value.

Each residual value E may be quantized using division with truncation by a divisor that is a power of 2 and using rounding with a rounding value that is 1 less than half the divisor.

```
If E<0 QE = (E−ROUND)/DIVISOR
Else QE = (E+ROUND)/DIVISOR
// the "/" operator is div with truncation as in C
```

Where:

```
DIVISOR=2**qLevel=1<<qLevel

ROUND=DIVISOR/2−1
```

The value of qLevel may be different for luma and chroma and is determined by the rate control (RC) function.

MPP quantized residuals may be checked to ensure that their sizes do not exceed a threshold such as maxBpc−qLevel, where qLevel is the quantization level for the component type (luma or chroma) and maxVal is the maximum possible sample value for the component type. If an

MPP residual exceeds this size, the encoder may change the residual to the nearest residual with a size of maxBpc−q Level.

Inverse Quantization and Reconstruction

The encoder may follow the same process used in the decoder to arrive at the reconstructed pixel values. For pixels that are predicted using MMAP, BP, or MPP, the reconstructed sample value may be:

```
reconsample=CLAMP(predSample+
    (quantized_residual<<qLevel),0,maxVal);
```

where predSample is the predicted sample value, quantized_residual is the quantized residual, qLevel is the quantization level for the component type (luma or chroma), and maxVal is the maximum possible sample value for the component type.

Flatness QP Override

FIG. **22** shows an example **2200** of original pixels used for encoder flatness checks. Encoders generate a "flatness signal" if upcoming input pixels are relatively flat to allow the QP to drop quickly. The encoder algorithm to determine the flatness bits in the syntax is described below, as is the algorithm that both the encoder and decoder follow to modify the QP.

Encoder Flatness Decision

A set of 4 consecutive groups is called a supergroup. The encoder examines each supergroup before it is encoded in order to determine which, if any, of the groups are "flat". The first supergroup starts with the 2nd group in the slice as shown in FIG. **22**. Supergroups may be defined consecutively within the slice. A supergroup that includes the last group of a line may wrap around to include groups on the subsequent line.

The flatness determination may be done for each group within the supergroup independently and includes a determination of the "flatness type" (e.g., either somewhat flat or very flat) for each group. Two flatness checks may be performed, both using pixels from the original, uncompressed image.

Flatness check 1 determines the MAX and MIN value among the samples shown in FIG. **22** for each component. A value of flatQLevel is determined for each component:

```
flatQLevel=MapQpToQlevel(MAX(0,masterQp−4));
```

The masterQp value that is used is the one that is used for rate control for the 2nd group to the left of the supergroup that is being tested. MapQptoQlevel maps the masterQP value to qLevelY (luma) and qLevelC (chroma) values that are used for both luma and chroma. For example, a masterQP value of 0 may map to qLevelC and qLevelY values of 0, values 1 and 2 may map to qLevelC values of 1 and 2 respectively, and successive unit increases in masterQP may map to unit increases alternating between qLevelY and qLevelC.

If the MAX−MIN for any component is greater than (2<<(bits_per_component−8)), the check for very flat fails for flatness check 1; otherwise, it passes. If the MAX−MIN for any component is greater than QuantDivisor [flatQLevel], the check for somewhat flat fails for flatness check 1; otherwise, it passes.

If flatness check 1 indicates that the group is either somewhat flat or very flat, that result is the final result that is used for the group. If both fail, flatness check 2 is performed over the 6 pixels indicated in FIG. **22**. The same comparisons are done as in flatness check 1, except that the MAX and MIN are computed over 6 samples rather than 4. The final result of flatness check 2 is then used as the final result for the group.

For a given supergroup, there are then four flatness indications of either not flat, somewhat flat, or very flat. The value of prevIsFlat is initialized to 1 if the previous supergroup had a flatness indication; otherwise it is initialized to 0. The following algorithm is used to distill the flatness information into a single flatness location and type:

```
Loop over four groups in supergroup {
    If !prevIsFlat && group is either very flat or somewhat
flat
        Current group and flatness type is signaled
    Else
        prevIsFlat = 0;
}
```

If no group is selected, no QP modification is made and flatness_flag for the supergroup is set to 0 in the entropy decoder. If a group is selected, the flatness_flag for the supergroup is set to 1, and the corresponding group is signaled as the first_flat group in the bit stream along with its associated flatness_type. The entropy encoder will only signal flatness_flag if the masterQp value is within the range of flatness_min_qp and flatness_max_qp, so no adjustment is made in the RC if the corresponding masterQp is out of range.

The encoder flatness searches do not span to the next line. If a group within a supergroup falls on the next line, it is not considered to be flat. However, the first group of a line may contain the next_flatness_flag syntax element assuming the syntax allows it at that point.

Flatness QP Adjustment

The encoder and decoder make the same QP adjustment for a group where a flatness indication has been made. The RC receives a flatness signal corresponding to a particular group within a supergroup that may be either "somewhat flat" or "very flat". It should be noted that if the current masterQp is less than 7<<(2*(bits_per_component−8)), the flatness indication may be assumed to be "somewhat flat".

For a "very flat" signal, the QP is adjusted as follows:

```
masterQp=1<<(2*(bits_per_component−8));
```

For a "somewhat flat" signal:

```
masterQp=MAX(stQp−4,0);
```

If there is no flatness signal for a particular group:

```
masterQp=stQp
```

If the flatness QP override modifies the masterQp, the modified masterQp is used as the starting point for the short-term rate control on the next RC cycle.

Buffer Model and Rate Control

In addition to, or as extensions of the implementations described above, e.g., with respect to FIG. **7**, some additional rate control techniques are next described. In some implementations, the encoders and decoders ensure that a complete image fits within a fixed number of bits. The virtual buffer model described employs a defined and controlled rate of bits per pixel and hence per unit time. In some implementations the total number of bits per picture may be larger than the product of the number of pixels times the bits per pixel, by up to the size of the buffer model. In some implementations, the encoders bound the entire coded picture size to the product of the number of pixels times the bits per pixel rate. In that respect, the entire picture may be coded such that it can be communicated over a number of line

times that is equal to the number of lines in the image, and there may be a fixed or bounded number of bits per pixel time within those lines.

One modification to the virtual buffer model described above is to increase the fullness of the buffer model, as used by the encoder and/or the decoder, by a fullness offset. The value of the fullness offset may be specified algorithmically. Delays from start of encoding to start of transmission, and from start of reception to start of decoding, are predetermined. For example, the initial transmission delay may be one line time, and the initial decoding delay may be one line time, while the buffer model size may be large enough to accommodate six line times at the specified transmission rate.

In this example, the fullness offset may have an initial value of 5*R*TI (5*the specified rate*one line time). This causes the encoder to ensure that the fullness of the buffer model does not exceed 1*R*TI, since the bottom 5*R*TI portion of the buffer model is not available for use due to the offset. The encoder may then reduce the offset value, by a predetermined amount for each predetermined number of pixels (e.g., once per pixel, or once per group of 3 pixels, or other amount). The predetermined amount of reduction in the offset value may be chosen such that the offset becomes 4*R*TI at the end of the first line of video, for instance. The result is that the maximum number of bits that the encoder might use to code the first line is 2*R*TI, and as a result transmission of the first line is assured of being completed within 2 line times, starting from the start of transmission. The decoder begins receiving data at the start of transmission plus an arbitrary delay. The decoder begins decoding one line time later, e.g., the initial decoding delay, and the decoder completes decoding the first line one line time after that. Therefore the decoder is assured of receiving all the data it needs to decode the first line by the time the data are needed.

The encoder may continue to reduce the fullness offset, e.g., by the same predetermined number of bits per pixel, until the offset value reaches zero. In this example, the offset reaches 0 at the end of the 5th line. The offset stays at zero until another time when the offset starts increasing. For example, the fullness offset may start increasing by the same predetermined number of bits pixel at the start of the line that is the 5th line from the end of the image, and it continues to do so until it has reached the value of 5*R*TI at the last pixel of the image. As a result of the fullness offset values, the encoder finishes coding the image with no more than 1*R*TI bits in the buffer model. This number of bits can be transmitted within 1*TI or less time. Since in this example the decoder started decoding the image 1*TI after it starts receiving the compressed data, it finishes decoding the last line of the image 1*TI after the last line time of data reception. As noted, all of the bits that encode the end of the image can be transmitted, and hence received, within 1*TI, and therefore all the bits of the image are received by the decoder by the time it needs them, including the last pixel of the image.

In this example, all of the bits of the image are transmitted within a number of line times equal to the number of lines of the image, at a predetermined rate R. The buffer model behavior facilitates high quality coding of the entire image, with significant rate control freedom at the first and last lines, and even more rate control freedom for the other lines. In this example, where the size of the buffer model is 6*R*TI, for all but the first 5 and last 5 lines of the image, the encoder has the freedom to utilize the full 6*R*TI size of the buffer model for efficient coding of image content.

The same technique may be applied to regions of an image. For example, it may be desirable for the encoder to partition an image into a fixed number of slices. Each slice may have substantially the same number of lines. For example, in an image with 1080 lines, there may be 8 equally sized slices of 135 lines each. The technique disclosed here may be used to ensure that each slice is fully communicated in 135 line times. In the explanation above, the start of a slice may substitute for the start of an image, and the end of a slice may substitute for the end of an image.

The buffer model for rate control converts a varying number of bits used to code each group or other set of samples into a specified constant bit rate. As each group is coded, the number of bits used to code the group is added to the buffer fullness, and the number of bits that is to be transmitted per group is subtracted from the buffer fullness. The result is referred to as the buffer model fullness or simply bufferFullness. This buffer fullness is modified by a linear transformation, e.g., offset and scale, to produce a value that is referred to as the rcModelFullness. The transformation may allocate extra bits to the first line of each slice and fewer bits to other lines, and to bound the maximum number of bits in the encoder buffer at the end of each slice to whatever the specified bound is. Both the first line allocation and the end of slice bound are configurable.

The number of bits removed from the buffer model each group may vary slightly from one group to the next in CBR operation, since the specified number of bits per group may include a fractional component. The bits_per_pixel rate may be specified using 4 fractional bits, for example, giving a resolution of 1/16 bit per pixel. If the specified number of bits per group is an integer, the number of bits removed from the buffer model every group is equal to the specified integer. If the fractional component is not zero, the fractional residual resulting from removing an integer number of bits each group is retained and applied to the next group.

In one implementation, the rcModelFullness may be defined to use negative values, where the empty state is represented by a value of −rc_model_size and the full state is represented by a value of 0. The RC algorithm may be designed to maintain the rcModelFullness value between empty (e.g. −rc_model_size) and full (e.g. 0). The offset value, rcXformOffset, and scale value, rcXformScale, are designed to convert the actual buffer fullness, bufferFullness, which is always non-negative, into a rate control buffer model fullness, rcModelFullness. The reason the empty level is numerically negative and the full level is 0 relates to the way the linear transformation is designed, as described below.

The RC algorithm selects a quantization parameter (QP) dynamically to both maintain the rcModelFullness within its valid range and to optimize subjective quality. In general, the RC seeks to code each group with approximately a target number of bits, while the number of bits spent coding each individual group can vary significantly. This behavior allows unexpectedly difficult image features to be coded efficiently while also coding smooth areas with very high accuracy; this helps maintain approximately equal subjective quality across the image without wasting bits.

The overall structure of the rate control technique **2300** is shown in FIG. **23**. The overall technique includes a buffer level tracker **2350**, linear transformation **2352**, parameter selection **2354**, and QP adjustment **2356**. Each is described in detail below.

The rate control techniques described above are very good for steady state operation, such as when the encoder is coding entire images. Slice rate control techniques described

below address coding an initial part of an image, such as the first line of a slice, while meeting the bound on the total number of bits per slice to num_pixels*bpp (bits per pixel rate).

One adaptation to the rate control techniques described above is keeping the total number of bits to be less than or equal to the product of a specified number of pixels (num_pixels) times bpp. The techniques may do so, in one implementation, by bounding the number of bits in the encoder's buffer when the last pixel is coded and delaying the start of transmission of each picture or slice according to this bound. A slice may be one or more lines high, where each line is typically one pixel high. Coding of the initial portion (e.g., the first line) of each slice or picture is jointly optimized with the coding of the remaining lines of the slice. A "slice" may include the case where a whole picture is one slice. The techniques allocate additional bits for the first line, in accordance with the unavailability of prediction information at the first line, and allocating accordingly fewer bits for all other lines in the slice.

The number of additional bits that the encoder may allocate for the first line depends on the number of bits for achieving the desired subjective quality level at the first line of each slice, and on the content being coded. The encoder may balance this number against the reduced number of bits that are allocated to the remaining lines in order to meet the constraint that total_bits<=num_pixels*bpp.

Image content to be compressed varies widely. For determining the values of the numbers of bits on the first and other lines, one may examine representative images, such as images that are considered difficult to encode. Empirical evidence indicates that, for example, for 12 bpp (bits per pixel) constant bit rate coding, pixels in the first line should be allocated on average an additional 5 bpp each. That is, the average number of bits used to code pixels in the first line may be approximately 17 bpp. However, the stream rate remains 12 bpp in CBR (constant bit rate) coding. The CBR rate refers to a rate of bits that exit a rate buffer, while the numbers of bits used to code each pixel enter the rate buffer. For images that are less challenging to encode, sufficient quality may be obtained with fewer bits on average spent coding each pixel. For images that are not especially difficult to encode well, it may be desirable to encode them losslessly, or at least with reduced quantization.

The encoder may enforce an upper bound, i.e. a maximum number of bits in the encoder buffer at the end of each slice regardless of the difficulty of coding each image. The maximum number of bits that may be permitted in the encoder buffer at the end of each slice may be determined empirically. Experiments have shown that 4 kb (4,096 bits) is a suitable value, but other values may be chosen depending on the application, for example 2 kb or 8 kb.

In one implementation, a rate control adaptation function utilizes a transformation of an encoder buffer model fullness to form a rate control (RC) buffer model fullness. These two values may be referred to as "actual fullness" and "RC fullness" respectively. The transformation may be a linear transformation, such as multiplication by a scale value and addition of an offset value, or it may be a non-linear transformation.

The RC fullness influences the quantization step or parameter (QP). Specifically, as the RC fullness increases, the QP may also increase, and the number of bits that code individual pixels is generally reduced. As the RC fullness decreases, the QP also decreases, and the number of bits that code individual pixels generally increases. The encoder may implement RC fullness ranges, and for any given range, there may be a minimum QP and a maximum QP. The RC control techniques below allocate additional bits for each pixel to the first line to help avoid image artifacts, particularly given that no prior prediction information is available for the first line to help with the encoding of the first line. The techniques may allocate a pre-determined number of additional bits for each pixel for the first line, leading to better coding consistency for the first line of a slice. The additional bit per each pixel budget may be taken away for subsequent lines.

In one implementation, the value of RC fullness may be calculated as RC fullness=(actual fullness+offset)*scale.

Buffer Level Tracker

FIG. **24** shows an example buffer level tracker **2350**.

The codedGroupSize input **2402** is an output from the entropy encoder or entropy decoder that indicates how many bits were used to code the previous group. The bitsPerGroup input **2404** is the number of bits allocated for each group, which may vary by +/−1 if bits_per_pixel contains nonzero fractional bits:

```
bpgFracAccum += (3 * bits_per_pixel) & 0xf;    // 4 fractional
bits
if(groupCount < initial_enc_delay)
    bitsPerGroup = 0;
else
    bitsPerGroup = floor(3 * bits_per_pixel) + (bpgFracAccum
>> 4);
bpgFracAccum &= 0xf;
```

If vbr_enable is equal to 1, the bufferFullness output **2406** is clamped at 0 if the final modified value would be less than 0. In this case, the forceMpp output **2408** may always be 0.

If vbr_enable is equal to 0, the bit stuffing detection logic **2410** checks if the next group could potentially cause an underflow condition (e.g., resulting in a bufferFullness that is less than zero). If so, the forceMpp output **2408** is set to 1, which indicates to the entropy encoder to use MPP mode in order to guarantee a minimum bit rate or to prevent buffer underflow. The forceMpp output **2408** may be determined as follows:

$$forceMpp=(groupCount>initial\_enc\_delay)\&\&(bufferFullness<ceil(bits\_per\_pixel*3)-3);$$

where groupCount is a counter that starts each slice at 0 and increments every group. The register **2412** stores the current value of the buffer fullness.

FIG. **25** shows an example of encoder logic **2500** that may implement rate control. The encoder logic **2500** implements a buffer model **2502**, transformation logic **2504** (which may implement the transformation logic **2352**), and quantization adjustment **2506** logic (which may implement the logic **2354** and **2356**, as examples). The encoder logic **2500** also includes an offset value generator **2512** that produces offset values **2508**, and a scale value generator **2514** that generates scale values **2510**. The implementation may be in hardware, software, or both, and is described in further detail below.

For illustration of the rate control, assume that the bpp rate is 12 bpp, the first line of each slice should be allocated 17 bpp, and the slice is 8 lines high. The total number of bits per slice is 12 bpp*slice_width*8 lines. With 17 bpp allocated to the first line, the budget for the remaining lines is (12 bpp*8 lines−17 bpp*1 line)/(8−1) lines, or approximately 11.29 bpp. Assume the maximum number of bits permitted in the buffer at the end of each slice=4 kb (kilobits). The number of pixel times corresponding to transmission of 4 kb is ceiling (4096 b/12 bpp)=342 pixel

times. Therefore the initial transmission delay is set to 342 pixel times. In other words, at the start of each slice no data are transmitted for the first 342 pixel times, and thereafter data are transmitted at 12 bpp for a number of pixel times equal to the number of pixels in the slice, including 342 pixel times after the last pixel is coded.

For the first 342 pixels of the first line, data accumulate in the encoder buffer at a rate of approximately 17 bpp, for a total of approximately 342*17=5,814 bits. Assuming an image width of 1920 pixels, there are 1920−342=1,578 additional pixels in the first line. As these pixels are coded, the number of bits used to code each pixel is expected to be on average 17 bpp, while the transmission rate is 12 bpp, hence coded data accumulate in the encoder buffer at rate of approximately 5 bpp. This additional accumulation is approximately 1578*5=7,890 bits. In this case the total accumulation of bits in the encoder buffer at the end of the first line is approximately 5,814+7,890=13,704 bits.

The encoder may generate values for the offset value **2508** using an offset value generator **2512**. The generator may produce offset values that follow a trajectory such that the sum of actual fullness+offset has desired values at various points in the coding of a slice or picture. For example, continuing with the example above, the expected actual buffer fullness after coding the first 342 pixels of a slice may be 5814 bits, and the desired RC fullness may be 2048 bits, hence the offset value may be 2048−5814=−3766 immediately after coding the 342nd pixel. The offset value before coding the first pixel may be 0. The expected actual fullness after coding the first full line may be 13704 bits. If the desired RC fullness is 2 kb at that point, the offset value may be 2048−13704=−11656. The offset value at the end of the slice may be 0. The offset value may progress linearly from the first specified value to the 2nd and then to subsequent specified values. That is, the offset value may be piecewise linear between inflection points, e.g., in FIG. **26**, from 0 to −3766, from −3766 to −11,656, and from −11,656 to zero. Note that the example above uses positive rate control thresholds (see FIG. **28**), and that the offset values may be negative or positive at any given point along the trajectory of the offset value. In other implementations, the rate control thresholds may be negative valued thresholds. In that case, the offset values may be strictly negative values, because actual fullness is non-negative, and a negative offset value would be used to bring the transformed fullness value down into a negative threshold range.

The encoder logic **2500** may generate values for the scale value **2510** using a scale value generator **2514**. The scale value generator **2514** may be designed to produce a scale value that follows a specified trajectory. For example, the value of scale may be 1 at the start of a slice and remain 1 until a certain pixel within the slice, and then linearly progress to another value such as 2 at the end of the slice. In the example of FIG. **24**, at the end of a slice the offset value is 0 and the scale value is 2. As a result, an actual fullness value of 4 k results in a RC fullness value of 8 k. This may be desirable if a design goal is to bound the maximum number of bits in the encoder buffer to 4 k, while the RC algorithm utilizes a control algorithm that bounds the RC fullness to 8 k. As with the offset value **2508**, the scale value **2510** may be piecewise linear between inflection points, e.g., in FIG. **26**, remaining at 1.0 for a time, and then linearly increasing from 1.0 to 2.0. The bit trajectories may differ between implementations. For example, the bit trajectories may try to keep the bottom of the effective rate control thresholds at approximately zero at the beginning and end of a slice. Further, approximations to such bit trajectories may be employed to reduce implementation complexity or for other reasons.

FIG. **26** shows an example of bit trajectories **2600** over time, as described above, including the scale parameter **2510** and offset **2508**. FIG. **26** also illustrates the effect of the parameters on RC fullness **2602**, and also shows the expected maximum actual buffer fullness **2604**. As shown, 17 bpp (12 bpp nominal plus 5 extra bpp) are allocated for the first 342 pixels, and 5 bpp are allocated for the remainder of the line. During time **2650**, bits accumulate in the buffer for the transmit delay of 342 pixels, and during time **2652** leave the buffer at 12 bpp thereafter. With 17 bpp allocated for the first line, 5 bpp accumulate in the buffer during time **2652**. After the first line, during time **2654**, the allocated bpp drops below 12, and the actual buffer fullness drops accordingly, as bits continue to leave the buffer at 12 bpp. At the end of the slice 2 kb actually remain in the buffer, but due to the scale value of 2.0, the RC fullness is 4 kb. The number of nominal bpp and the additional bit budget for the first line may vary widely. For example, the number of nominal bpp may range between 6 and 24.

The RC techniques permit a non-zero actual buffer fullness at the end of the slice. This may help avoid heavy quantization at the end of the slice, which leads to visual artifacts. The RC techniques allow the rate buffer to track to where it would normally go, responsive to the image content. Note that the scale value **2510** is applied to the buffer fullness. Accordingly, a scale value of 2.0 effectively drops the nominal buffer range (e.g., of 8 kb) to a smaller range (e.g., 4 kb). That is, the encoder may apply the scale value to decrease the effective buffer range when the encoder wants the real buffer fullness to track within a smaller range.

Alternatively, the offset value generator **2512** may be configured to generate values of offset that reach, for example, 4 k at the end of a slice. For example, an actual fullness at the end of a slice of 4 k added to an offset value of 4 k produces an RC fullness value of 8 k at the end of the slice.

Alternatively, the offset value **2508** may follow a trajectory that ends with a negative value such as −4 k. In this example the RC algorithm may have an effective range of RC fullness from −8 k to 0, where −8 k may be correspond to an empty buffer model and 0 may correspond to a full buffer model. The scale value may follow many different trajectories. For example, one trajectory has a value greater than or equal to 1 from the start of the slice, then decreasing to, e.g., a value equal to 1 while encoding the interior of the slice, then increasing to a scale value to greater than or equal to 1 at the end of the slice.

FIG. **27** shows an example **2700** of offset fullness and actual fullness in relation to the scale parameter **2510**, for different types of content. In particular, the example **2700** illustrates offset fullness and actual fullness for what is considered worst case content, hard content, easy content, and moderate content. The references in FIG. **27** to "WC" indicate "worst case" scenarios.

The linear transformation by the scale value **2510** and offset value **2508** by the transformation logic **2504** manages the buffer fullness over the course of the slice. It has three main functions: 1) keep the quality constant during the slice, including the initial delay; 2) allocate extra bits for the first line of each slice; and 3) ensure that the slice is coded within the correct number of bits by constraining the final encoder buffer fullness.

FIG. **27** shows the range compression caused by the scale value **2510**. Starting on the left, with a range of actual buffer fullness 0 to 8 kb, and on the right, ending the slice with a

range of 0 to 4 kb. FIG. **27** shows the bit trajectories to the right of the end of the first line **2702**. Note that the actual fullness for 'easy' content does not go below zero, but that additional bits may be generated to keep the actual fullness above zero.

In the lower section of FIG. **27**, the buffer model fullness ranges of 0 to −8 k are shown. The lower section shows what happens to the fullness after the scale and offset are applied, and how it falls within the ranges.

In one implementation, the linear transformation logic **2352** or **2504** implements the following, where the scale value **2510** is referred to as rcXformScale and the offset value **2508** is referred to as rcXformOffset:

$$rc\text{ModelFullness}=(rc\text{XformScale}*(\text{bufferFullness}+rc\text{XformOffset}))>>3$$

The rcXformOffset is designed to perform the functions listed above. The rcXformScale factor is applied throughout the slice to convert a reduced actual buffer fullness range to a complete buffer model fullness range at the end of a slice, to have a certain effect on the range conversion at the start of a slice, and to gradually change the conversion over the course of a slice, for example starting after the first line.

The encoder may choose a range of values of rcXformOffset to be negative in order to produce a negative range of rcModelFullness. This is done so that a coarse resolution of the rcXformScale factor has minimum effect on the value of rcModelFullness when it is nearly full, since the error term resulting from coarse quantization times a value near zero results in an error that is near zero. The rcXformScale factor quantization error is instead shifted to the empty end of the rcModelFullness range, where it has an insignificant effect.

The rcXformOffset value starts each slice at a known initial value initial_offset−rc_model_size. The rcXformOffset modification per group includes of the superposition of several things:

In one implementation, during the initial delay, the rcXformOffset decreases at a rate of (bits_per_pixel*3) bits per group.

During the entire slice, the rcXformOffset increases at a rate of slice_bpg_offset.

During the first line of a slice, the rcXformOffset decreases at a rate of first_line_bpg_offset bits per group.

During the non-first lines of a slice, the offset increases at a rate of nfl_bpg_offset bits per group.

The rcXformOffset value may be prevented from exceeding final_offset−rc_model_size during non-first lines of a slice, although this limit is unlikely to be enforced until near the end of a slice.

The rcXformOffset value is tracked with a precision of, for example, 11 fractional bits. So the per-group adjustments, such as slice_bpg_offset or nfl_bpg_offset, are specified with 11 fractional bits of precision. At the beginning of a slice, the initial rcXformScale value is set to initial_scale_value. Accordingly, the initial scale factor may be greater than 1 at the beginning of a slice. At the beginning of a slice, the rcXformScale factor decreases by 1 every scale_decrement_interval groups until it reaches unity scaling.

On the last line of a slice, the rcXformScale factor ramps up smoothly from, e.g., 8 (in units of ⅛) by incrementing by, e.g., 1 every scale_increment_interval groups.

The net effect of the rcXformOffset and rcXformScale is to allow the buffer fullness to grow according to an allocation of extra bits in the first line and a specified initial transmission delay, to smoothly ramp down the maximum fullness from the end of the first line until the end of the slice, and to ensure that the number of bits in the buffer at the end of the slice does not exceed initial_enc_delay*3*bits_per_pixel−the maximum number of padding bits that could be generated by the substream multiplexing process.

Long Term Parameter Selection

As noted above, long term parameter selection logic **2354** is included in the encoder. In the long term parameter selection logic **2354**, the value of rcModelFullness may be classified as being in one of a number of ranges, for example the ranges **2800** shown in FIG. **28**. The set of ranges is determined by a set of thresholds. There may be, for example, 15 ranges that are defined by 14 thresholds (rc_buf_thresh) and the rc_model_size. For each range, there may be a minimum quant value (rc_min_qp), a maximum quant value (rc_max_qp), and an offset (rc_bpg_offset) that adjusts the target bits per group.

The rc_min_qp and rc_max_qp values for each range are configured such that when the RC buffer fullness is at or near empty, the RC sets the masterQp value either to 0 or near zero, and as the RC buffer fullness approaches full, the RC increases the masterQp value, eventually reaching a point where it sets the masterQp to the maximum valid value when the RC buffer fullness is nearly full. The target number of bits per group is greatest when the RC fullness is empty and least when the RC fullness is full.

The rc_model_fullness is compared to a number of thresholds to determine which one of 15 ranges it is in. Each range has an associated rc_min_qp, rc_max_qp, and rc_bpg_offset that are used for the short-term rate control. In one implementation, the encoder uses threshold values from −rc_model_size to 0, and these values can be found by subtracting the rc_model_size from a set of positively defined thresholds. The 6 LSB's of each threshold may be assumed to be zero to facilitate an efficient look-up table implementation for the threshold comparison function.

The values minQp, maxQp, and bpgOffset at each range of buffer model fullness are loaded with the rc_min_qp[ ], rc_max_qp[ ], and rc_bpg_offset[ ] values that correspond to the range corresponding to rcModelFullness.

Short Term Parameter Selection

As noted above, short term parameter selection logic **2356** is included in the encoder. The short term parameter selection logic **2356** makes adjustments to the QP, and may use information from the entropy encoder in order to make final adjustments to the QP.

The short term parameter selection logic **2356** may implement the short-term rate control logic shown in FIG. **29** and the QP increment logic shown in FIG. **30**. The parameter minQP refers to the minimum QP value permitted for a given range. The value of the previous QP, prevQp, is the most recent master QP value, masterQp, that was generated for the previous group. The masterQp that was used before that is referred to as prev2Qp in FIG. **30**.

The logic **2900** determines a bits per group (BPG) target, tgtMinusOffset, and a tgtPlusOffset (**2902**). Depending on the results of the tests **2904**, **2906**, and **2908**, the short term QP (stQP) is changed. Specifically stQP may change to: the maximum of the previous QP minus 1 and the minimum QP divided by 2 (**2910**), the maximum of the previous QP minus 1 and the minimum QP (**2912**), an incremented value (**2914**), or may remain at the previous QP (**2916**).

FIG. **30** shows the example QP increment logic **3000** that may generate the incremented value, e.g., in connection with FIG. **29** (**2914**). The logic **3000** sets the current QP to the maximum of the minimum QP and the previous QP (**3002**). Depending on the results of the tests **3004**, **3006**, **3008**,

3010, and 3012, stQP is set according to one of two options. In the first option, the stQP is set to the current QP (3014). In the second option, the stQP is set to the minimum of the maximum QP and the current QP plus an increment amount (3016). The increment amount, incrAmount, may be determined as described in the following paragraphs.

The value of rcXformBpgOffset is an offset that is positive for the first line in each slice and negative for all other lines in the slice, which is calculated internally:

```
if ( first line of slice )
  rcXformBpgOffset = first_line_bpg_offset;
else
    rcXformBpgOffset = -floor(nfl_bpg_offset);
if ( groupCount >= initial_enc_delay)
    rcXformBpgOffset -= floor(slice_bpg_offset);
```

The target number of bits for each group, e.g., the target activity level used by the rate control, is called rcTgtBitsGroup:

$$rcTgtBitsGroup=round(3*bits\_per\_pixel)+bpgOffset+rcXformBpgOffset$$

In addition to responding to the rcModelFullness, the RC adjusts the QP according to a measure of the activity of the image, using values from the entropy coding called rcSizeGroup and codedGroupSize, which are rough measures of the activity of the group preceding the current group. The rate control calculates high and low bits per group thresholds:

$$tgtMinusOffset=rcTgtBitsGroup-rc\_tgt\_offset\_lo$$

$$tgtPlusOffset=rcTgtBitsGroup+rc\_tgt\_offset\_hi$$

The codedGroupSize and rcSizeGroup values are compared to tgtMinusOffset and tgtPlusOffset to determine whether the activity of the local region of the image is within the expected range, below the expected range or greater than the expected range. The value for rcSizeGroup is also compared to the constant 3 which represents the minimum possible number of bits per group. Based on these comparisons, the RC increases or decreases QP or leaves QP unchanged subject to the min and max QP bounds that apply to each range.

If the rcModelFullness falls in the top-most range, the QP may be automatically set to rc_max_qp for that range to avoid overflowing the buffer. There are three other parameters that are shown FIG. 30: rc_edge_factor, rc_quant_incr_limit1, and rc_quant_incr_limit0.

A description of the parameters used above follows:

rcXformBpgOffset—an internal variable that represents a bits per group offset selected for different lines of a slice, determined as noted above.

first_line_bpg_offset—This value specifies the number of additional bits that are allocated for each group on the first line of a slice.

nfl_bpg_offset—This value specifies the number of bits (including fractional bits) that are deallocated for each group for groups after the first line of a slice.

slice_bpg_offset—This value specifies the number of bits (including fractional bits) that are deallocated for all groups in order to enforce the slice constraint (e.g., that the final fullness cannot exceed the initial encoder delay*bits per group).

initial_offset—This value specifies the initial value for rcXformOffset, which is, for example, initial_offset−rc_model_size at the start of a slice.

final_offset—This value specifies the maximum end-of-slice value for rcXformOffset, which is, for example, final_offset−rc_model_size

rc_edge_factor—This value may be compared to the ratio of current activity to previous activity in order to determine the presence of an "edge", which in turn determines whether or not the QP is incremented in the short-term rate control.

rc_quant_incr_limit0—This value may be a QP threshold that is used in the short-term rate control.

rc_quant_incr_limit1—This value is a QP threshold that may be used in the short-term rate control.

rc_tgt_offset_hi—This value specifies the upper end of the range of variability around the target bits per group that is allowed by the short-term rate control.

rc_tgt_offset_lo—This value specifies the lower end of the range of variability around the target bits per group that is allowed by the short-term rate control.

The increment to the QP (incrAmount) may be determined according to:

$$incrAmount=(codedGroupSize-rcTgtBitsGroup)>>1;$$

The resulting QP is called stQP, which may be modified by the flatness QP override logic that was described above with regard to FIG. 22.

The encoder maps the masterQp value to qLevelY and qLevelC values that are used for both luma and chroma. The encoder may implement a wide variety of mappings, once of which is shown in the table below.

	8bpc		10bpc		12bpc	
masterQp	qLevelY	qLevelC	qLevelY	qLevelC	qLevelY	qLevelC
0	0	0	0	0	0	0
1	0	1	0	1	0	1
2	0	2	0	2	0	2
3	1	2	1	2	1	2
4	1	3	1	3	1	3
5	2	3	2	3	2	3
6	2	4	2	4	2	4
7	3	4	3	4	3	4
8	3	5	3	5	3	5
9	4	5	4	5	4	5
10	4	6	4	6	4	6
11	5	6	5	6	5	6
12	5	7	5	7	5	7
13	5	8	6	7	6	7
14	6	8	6	8	6	8
15	7	8	7	8	7	8

-continued

	8bpc		10bpc		12bpc	
masterQp	qLevelY	qLevelC	qLevelY	qLevelC	qLevelY	qLevelC
16			7	9	7	9
17			7	10	8	9
18			8	10	8	10
19			9	10	9	10
20					9	11
21					9	12
22					10	12
23					11	12

Returning to the offset and scale parameters noted above with respect to FIG. **25**, and giving some specific examples, the offset value generator **2512** and the scale value generator **2514** may produce coarse approximations of the values described as linear trajectories. For example, a differential value may be added to an accumulator for every certain number of pixels. The scale and offset values and the operations that use them may have a specified accuracy and resolution.

As a specific example, it may be desirable to obtain approximately 2 kb of RC fullness throughout the coding of a slice. The encoder may apply, as noted above, a linear transformation comprising an offset and a scale factor to the actual fullness to produce the RC fullness. Throughout the first line, the scale factor has a value of 1.0. At the first pixel of the first line, the offset value=0 and the scale value=1. While it may be desirable to utilize a positive offset value at the first pixel in order to obtain an RC fullness value of 2 kb, this may not be desirable with less difficult content that could potentially be coded without quantization loss, hence in this example the initial offset value is set to 0.

At the first pixel, the RC fullness equals the actual fullness since the scale factor is 1 and the offset is 0. As the leading pixel set (e.g., the first 342 pixels) are coded, the actual fullness increases at a rate of approximately 17 bpp for difficult images. To achieve an RC fullness that is 2 kb at the end of the leading pixel set, the offset value decreases linearly from 0 to a value of −(342*17)+2048=−3,766 at the 342nd pixel. That is, for an actual fullness of 342*17=5,814 bits, an offset of −3,766 is applied, resulting in an RC fullness of 5,814−3,766=2,048 bits. As the remaining 1920−342=1578 pixels of the first line are coded, again the RC fullness is maintained at approximately 2 kb by having the offset value decrease linearly from −3,766 at the 342nd pixel at a rate of 5 bpp for 1578 pixels, resulting in a value of −3766−(5*1578)=−11,656 at the end of the first line. At the end of the first line, the number of bits used to code the first line is expected to be approximately 1920*17=32,640, the actual fullness is expected to be approximately 342*17+1578*5=13,704 bits and the resulting RC fullness is 13,704−11,656=2,048 bits.

After coding the end of the first line, e.g., at the start of coding the 2nd line, the offset value begins to increase linearly from its initial value of −11,656 to a final value of 0 at the last pixel of the slice. Again, while it might be desirable to set the offset to a positive value at the end of the slice for purposes of coding a difficult image, it may be preferable to maintain the offset value at least than or equal to 0, to enable lossless coding of images or slices which could potentially be coded losslessly at the available bit rate. If the scale factor were maintained at 1.0 throughout the slice and the actual fullness were 2 kb at the end of the slice, the offset of 0 would result in an RC fullness value of 2 kb. However, the scale factor increases linearly from 1.0 at the end of the first line to 2.0 at the end of the slice. As a result, an actual fullness of 2 kb produces an RC fullness of 4 kb. Since in this example the RC has a span of 8 kb, the maximum RC fullness that the RC permits is 8 kb; this corresponds to an actual fullness of 4 kb, which corresponds to the maximum actual allowed fullness at the end of the slice.

The combination of the increasing offset and the increasing scale factor from the start of the 2nd line until the end of the slice decreases the bit budget available for coding pixels from the 2nd line through the end of the slice. If the content being coded results in an actual buffer fullness of 4 kb at the last pixel of the slice, the bit budget for the 2nd through 8th lines is the total number of bits minus the expected number of bits used to code the first line divided by the remaining number of pixels, (1920*8*12−1920*17)/(1920*7) or approximately 11.29 bpp, as noted previously. The adjustment of the offset value results in a reduction in the bpp budget of 11,656 bits/(1920 pixels/line*7 lines) or approximately 0.867 bpp, resulting in a net bpp budget of 12−0.867 or approximately 11.133 bpp, if the actual fullness at the end of the slice is the same as it is at the end of the first line. However, if the actual fullness at the end of the slice is 2 kb more than it is at the end of the first line, e.g., 4 kb at the end of the slice vs. 2 kb at the end of the first line, the reduction in bit budget is (11,656−2048)/(1920*7) approximately 0.715 bpp, for a net of 12 bpp−0.715 bpp or approximately 11.29 bpp. The increase in the scale factor to 2.0 allows the maximum number of bits at the end of the slice to be bounded by the RC to 4 kb, while the RC range spans 8 kb. For content that does not produce a large number of bits at or near the end of the slice such that the actual fullness would be small in the absence of a scale factor, the scale factor may have little effect.

The encoder may be configured to insert stuffing bits after producing the last bits that code pixels in a slice when the total number of bits used to code a slice is less than the target number, thereby producing exactly the target number of bits. The target number of bits may be the product of the number of pixels in a slice times the bits/pixel rate.

In the encoder, offset value generator **2512** may compute offset values by adding an incremental value to an accumulator each pixel, Group or other interval. For instance, there may be three increment values, a first value for the first transmission delay portion of the slice, a second value for the remainder of the first line, and a third value for the remainder of the slice. If the offset is incremented every 3 pixels (one Group), there are 114 groups in the first 342 pixels, corresponding to the initial transmission delay. The values given in the example here result in a first increment value of approximately −33.035. The accuracy of the increment may be chosen to be sufficient such that the value of

the offset at the end of the initial transmission delay is close enough to the desired value of −3766; it does not need to be exact. For example, if an error of 1% or approximately 38 is permitted, the increment value may differ from the ideal increment value by 38/114=0.333. Using binary arithmetic, the increment may be specified with 1 fractional bit such that the maximum error is 0.25. Thus the first increment value may be −33.00. The resulting offset value at the end of the transmission delay interval is then −342*33.0=−3,762. The second increment value in this example is [−11,656−(−3762)]/(1920/3−114) or approximately −15.008. If again we allow a maximum error of 1%, 116, the increment error may be as high as 116/526 Groups=0.22. The increment may be specified with 2 fractional bits such that the maximum error is 0.125. Thus the 2nd increment value may be −15.00. The resulting offset value at the end of the first line is then −3,762−526*15=11,652. The 3rd increment value should be 11,652/(1920/3*7) or approximately 2.6009. If the allowed maximum error is for example 100, the increment error may be as high as 100/(1920/3*7)=100/4,480 or approximately 0.0223. The increment may be specified with 5 fractional bits such that the maximum error is 0.015625. The 3rd increment may be specified as 2.59375 in base 10, or 10.10011 in base 2. The resulting final offset value at the end of the slice is −11652+11620=−32. This is within the postulated acceptable limits.

The scale value generator **2514** may determine the scale value **2510** in a similar fashion. In one implementation, the scale factor **2510** may increase from 1.0 to 2.0 over the course of 7*1920/3=4480 Groups. The increment may be 1/4480 or approximately 0.223215E-3. If the maximum error of the final scale value in the negative direction is 0 and the maximum error in the positive direction is for example 2%, the maximum increment error is 0.02/4480 or approximately 4.464E-6. This implies specifying the increment such that its least significant bit corresponds to 2**−18, resulting in a maximum positive error of ~3.815E-6. The most significant bit needed for the increment corresponds to 2**−13, for a total of 6 significant bits. An adder accumulating such an increment every group may use 20 bits.

Alternatively the calculation of the scale factor **2510** may be updated less frequently, for example every 64 Groups. In this case there are 7*(1920/3)/64=70 increment steps. In this form, the increment value may be simply shifted left by 6 bits compared to the per-Group approach above, and the accumulating adder may have 20−6=14 bits.

Another alternative approach to incrementing the scale factor **2510** is to increment it by a simple constant such as 1 with an interval that is selected to produce the desired results. For example, an 8 bit counter could be used to calculate the values between 1.0 and 2.0. This counter could be incremented for example every 4480/256 Groups i.e. every 17.5 Groups, which may be closely approximated by incrementing every 17 or 18 Groups with the interval alternating each increment. Such an alternating interval may be implemented with a 5 bit counter and small amount of logic.

The same approach may be used for other specific design parameters. For example, a bit rate of 8 bpp and a first line allocation of 8+4=12 bpp. Many of the specific values in the example above are replaced with values derived from these parameters, and the operation may be substantially the same.

For utilization in a specific standard or product, the relevant parameters may either be specified in advance and built into the implementations, or alternatively one or more parameters may be calculated in software and loaded into an implementation. In one embodiment the encoder side of the system calculates the parameters and specifies them in a configuration header which is transmitted along with each picture, such that decoders may directly load and utilize the values in the headers, without requiring software interaction in the decoder side of the system.

FIG. **31** shows an example of substream demultiplexing logic **3100** in a decoder. In the demultiplexing logic **3100**, a rate buffer **3102** feeds a demultiplexer **3104**. The demultiplexer **3104** provides component (e.g., Y, Co, Cg) samples to the funnel shifters **3106**, **3108**, and **3110**. In turn, the outputs of the funnel shifters **3106**, **3108**, and **3110** provide data to the entropy decoders **3112**, **3114**, and **3116**. The rate control logic **3118** coordinates the operation of the entropy decoders **3112**, **3114**, and **3116**.

The demultiplexer **3104** receives requests from each funnel shifter (**3106**, **3108** or **3110**) that indicates that a mux word is needed. The request signal is sent if the current funnel shifter fullness minus the decoded syntax element size is less than the maximum syntax element size. Either 0, 1, 2, or 3 requests may occur for any given group time. If multiple requests are asserted in a given group time, the order of the mux words in a slice is muxWordY followed by muxWordCo followed by muxWordCg.

If vbr_enable is equal to 0, the demultiplexer flushes any zero-stuffing bits that were added at the end of a slice to pad the slice to a total compressed size of ceil (slice_width*slice_height*bits_per_pixel). If vbr_enable is set to 1, then no stuffing bits are removed from the end of the slice.

Entropy Decoding

The entropy decoders **3112-3116** parse the bits from the incoming bitstream after demultiplexing. The picture layer is demultiplexed to extract the slice layer bits for each slice. The substream demultiplexer demultiplexes the slice layer data into 3 substreams. The entropy decoder parses the substream layer.

Each group in the substream layer may be processed sequentially. Some groups have conditional bits at the beginning of the luma unit associated with flatness determination. Once each group has been processed, the entropy decoder sends the residual and ICH index data to the pixel reconstruction and ICH blocks. The entropy decoder outputs the total number of bits parsed for the entire group (codedGroupSize) and the number of bits that would have been used had the sizes been optimally predicted (rcSizeGroup) to the rate control.

After each group is processed, the resulting residuals and ICH selections are passed to the reconstruction and ICH blocks.

Each line may start on a group boundary. If the slice width is not evenly divisible by 3, the last group of each line may contain fewer than 3 pixels. However, the entropy decoders may still parse 3 residuals in P-mode and 3 history indices in ICH-mode. Although no pixel data is produced for pixels beyond the edge of the slice, the P-mode residuals are still used for the purposes of calculating the next predicted size.

If the input rate buffer overflows, the decoder may treat the overflow as an error condition. The decoder may count the bits as they are decoded, and may flag an error condition if the entropy decoder attempts to parse bits beyond the end of the slice data. The slice data length is either fixed (if vbr_enable is set to 0) or is variable and communicated to the decoder by the transport (if vbr_enable is set to 1).

Rate Control

The rate control logic **3118** may implement the same rate control as implemented in the encoder. The encoder and decoder rate control produce the same QP values at every group.

The decoder rate control logic **3118** may function as though it were encoder rate control logic. For each group, where the encoder encodes the group and adds the number of bits used to code the group to its buffer model fullness, the decoder adds the same number of bits to its buffer model fullness when it decodes the group. Both the encoder and decoder RC algorithms subtract the same number of bits when encoding or decoding the same group.

The decoder RC buffer model is the same as the encoder RC buffer model. However the operating context of a decoder is different from that of an encoder. The decoder has a rate buffer **3102**, which may be different than the encoder buffer model.

A bitstream (minus the PPS) to be decoded enters the decoder rate buffer **3102**, and the decoder removes bits from the rate buffer **3102** as the bits are decoded. This is opposite to the sense in which the RC buffer model operates. At the start of each slice, the decoder accumulates bits in its rate buffer for initial_dec_delay group times before starting to decode the slice. Once decoding begins, the RC function behaves the same as in the encoder, including the function of initial_enc_delay.

The flatness information is conveyed to the decoder RC via the entropy decoders **3112-3116**. The flatness information for a given supergroup is signaled in the previous supergroup to simplify the entropy decoding and timing. If the flatnessFlag for a given supergroup is 0, no QP adjustment is made. If the flatnessFlag is 1, the flatnessGroup signals which of the 4 groups requires the QP adjustment and flatnessType indicates whether the content is somewhat flat or very flat. If the flatnessType is not explicitly signaled in the bitstream because the QP was too low, the flatnessType is assumed to be 0 (somewhat flat). The adjustment is done in exactly the same manner noted above for flatness QP adjustment.

Line Storage

Like the encoder, the decoder may implement line storage. The line storage in the decoder may be similar to, or the same as, the line storage described above for the encoder.

Prediction and Reconstruction

The prediction and reconstruction functions in the decoder may match the corresponding encoder functions.

Prediction Types

The decoder prediction types may be the same as those in the encoder: MMAP, BP, and MPP.

Prediction Type Selection

The prediction type need not be explicitly signaled in the bitstream, so both the encoder and decoder may follow identical processes to determine which prediction type is used for each group. If a decoder supports block prediction, there may be logic to select between BP and MMAP; if the decoder does not support block prediction or bp_enable is set to 0 in the PPS, then BP is never selected and MMAP is used. If the decoder does not support block prediction and bp_enable is set to 1 in the PPS, the stream is not decodable, and the decoder shall handle the error in an appropriate manner.

Selection Between BP and MMAP

Encoders and decoders may perform the same algorithm to select between BP and MMAP.

Selection Between BP/MMAP and MPP

The selection between BP/MMAP and MPP may be signaled in the bitstream. The size used for DSU-VLC encoding determines whether MPP or BP/MMAP is used in the decoder. If the size is equal to the maxBpc−qLevel for some component, that component is predicted using MPP for all three samples in that group. Otherwise, BP or MMAP is used for that component for all three samples in the group.

FIG. **32** shows indexed color history (ICH) logic **3200** in a decoder. The decoder may have the same mapping of ICH values to pixels as the encoder, for each group. The decoder history buffer **3202** may be structured the same way as the encoder history. The decode process for updating the ICH may be the same as the encoder process for updating the ICH.

For each group, the entropy coding indicates whether ICH is selected or not. If ICH is selected, three history indices are provided by the entropy decoder as well. Both the encoder and decoder may maintain identical ICH state, so the update process may follow the process identified above.

Color Space Conversion

The display stream coding may utilize components that are labeled Y, Co, and Cg, or it may utilize components that are labeled Y, Cb and Cr. If the convert_rgb flag is equal to 0 in the current PPS, the decoder may produce YCbCr output without performing color space conversion. The Cb component may be mapped to the Co component label. The Cr component may be mapped to the Cg component label. In this case, the bit depth of the Cb/Co and Cr/Cg components may be equal to the Y component, whose bit depth is specified using the bits_per_component field in the current PPS.

If the convert_rgb flag is equal to 1 in the current PPS, the decoder performs color-space conversion from YCoCg to RGB. First, the Co and Cg values may be re-centered around 0:

$$cscCg = Cg - (1 \ll \text{bits\_per\_component})$$

$$cscCo = Co - (1 \ll \text{bits\_per\_component})$$

where bits_per_component is the number of bits of each of the R, G and B components, which is one less than the number of bits per component for the Co and Cg components.

The final CSC may be:

$$t = y - (cscCg \gg 1)$$

$$cscG = cscCg + t$$

$$cscB = t - (cscCo \gg 1)$$

$$cscR = cscCo + cscB$$

The final R, G, and B values may be range limited:

$$R = \text{CLAMP}(cscR, 0, \text{maxVal})$$

$$G = \text{CLAMP}(cscG, 0, \text{maxVal})$$

$$B = \text{CLAMP}(cscB, 0, \text{maxVal})$$

where $\text{maxVal} = ((1 \ll \text{bits\_per\_component}) - 1)$.

If a slice extends beyond the right edge of a picture, the resulting decoded pixels may be discarded. If a slice extends beyond the bottom edge of a picture, the resulting decoded pixels may be discarded.

Error Handling

If an error condition is detected, the decoder may output pixel data until the end of the slice. Such pixel data may have

any arbitrary value. The decoder may discard any compressed bits in the rate buffer remaining in the slice. The decoder may resume decoding with the next slice, and occurrence of an error in a slice need not affect decoding of any other slice.

The methods, devices, and logic described above may be implemented in many different ways in many different combinations of hardware, software or both hardware and software. For example, all or parts of the system may include circuitry in a controller, a microprocessor, or an application specific integrated circuit (ASIC), or may be implemented with discrete logic or components, or a combination of other types of analog or digital circuitry, combined on a single integrated circuit or distributed among multiple integrated circuits. All or part of the logic described above may be implemented as instructions for execution by a processor, controller, or other processing device and may be stored in a tangible or non-transitory machine-readable or computer-readable medium such as flash memory, random access memory (RAM) or read only memory (ROM), erasable programmable read only memory (EPROM) or other machine-readable medium such as a compact disc read only memory (CDROM), or magnetic or optical disk. Thus, a product, such as a computer program product, may include a storage medium and computer readable instructions stored on the medium, which when executed in an endpoint, computer system, or other device, cause the device to perform operations according to any of the description above.

The processing capability of the system may be distributed among multiple system components, such as among multiple processors and memories, optionally including multiple distributed processing systems. Parameters, databases, and other data structures may be separately stored and managed, may be incorporated into a single memory or database, may be logically and physically organized in many different ways, and may implemented in many ways, including data structures such as linked lists, hash tables, or implicit storage mechanisms. Programs may be parts (e.g., subroutines) of a single program, separate programs, distributed across several memories and processors, or implemented in many different ways, such as in a library, such as a shared library (e.g., a dynamic link library (DLL)). The DLL, for example, may store code that performs any of the system processing described above.

Various implementations have been specifically described. However, many other implementations are also possible.

What is claimed is:

1. A video-coding method comprising:
- obtaining a fullness value for a buffer;
- applying a transformation to the fullness value to obtain a transformed fullness, where applying the transformation comprising:
- applying a piecewise linear offset value to the fullness value; and
- applying a piecewise linear scale factor to the fullness value;
- making a quantization decision responsive to the transformed fullness; and
- coding data for a portion of an image in the buffer according to the quantization decision.

2. The video-coding method of claim **1**, where making the quantization decision comprises:
- determining a target number of bits per unit of data to be coded.

3. The video-coding method of claim **1**, where making the quantization decision comprises:
- allocating a selected number of bits per unit of data to be coded for a first image line; and
- allocating fewer than the selected number of bits per unit of data to be coded for a different image line subsequent to the first image line.

4. The video-coding method of claim **1**, where making the quantization decision comprises:
- allocating, for a first unit of data additional coding bits than are allocated for a second unit of data with more prediction information than the first unit of data.

5. The video-coding method of claim **1**, where coding comprises:
- coding a line of an image slice.

6. A video-coding system comprising:
- a buffer operable to store coding units to be coded;
- transformation circuitry configured to:
- obtain a fullness value for the buffer; and
- apply a transformation to the fullness value to obtain a transformed buffer fullness, where the transformation comprises:
- a piecewise linear offset value applied to the fullness value; and
- a piecewise linear scale factor applied to the fullness value;
- quantization adjustment circuitry configured to, responsive to the transformed buffer fullness:
- allocate a number of target bits for coding the coding units that varies responsive to an amount of prediction information that is available for the coding units; and
- coding circuitry configured to code data for a coding unit of an image in the buffer according to the number of target bits.

7. The video-coding system of claim **6**, where:
- the quantization adjustment circuitry comprises a mapping of transformed buffer fullness to a quantization output.

8. The video-coding system of claim **7**, where:
- the quantization output comprises:
- a minimum quantization parameter; and
- a maximum quantization parameter.

9. The video-coding system of claim **7**, where:
- the quantization adjustment circuitry comprises:
- long term quantization parameter selection circuitry configured to map the transformed buffer fullness to a first quantization output according to multiple transformed buffer fullness ranges defined by the long term quantization parameter selection circuitry.

10. The video-coding system of claim **9**, where:
- the quantization adjustment circuitry further comprises:
- short term quantization parameter adjustment circuitry configured to receive the quantization output from the long term quantization parameter selection circuitry, and responsively determine a different, second quantization output.

11. A video-coding system comprising:
- buffer level tracker circuitry comprising a buffer fullness output;
- linear transformation circuitry coupled with the buffer fullness output and comprising:
- an offset value generator comprising an offset value output comprising a piecewise linear offset value;
- a scale value generator comprising a scale value output comprising a piecewise linear scale value; and
- transformation circuitry configured to apply a linear transformation to the buffer fullness output responsive to the

offset value output and the scale value output, to obtain a modified buffer fullness output;

multiple stage quantization selection circuitry comprising:

a mapping of modified buffer fullness to a quantization parameter range; and

parameter adjustment circuitry configured to obtain a quantization parameter within the quantization parameter range responsive to a prior coding result and a prior value of the quantization parameter; and

coding circuitry configured to code data for a portion of an image in the buffer according to the quantization parameter.

12. The video-coding system of claim **11**, where:

the linear transformation is configured to cause additional bits per pixel to be allocated to an image line without prediction information, as compared to subsequent image lines with prediction information.

13. The video-coding method of claim **1**, further comprising where making the quantization decision comprises selecting a quantization output responsive to a mapping of the transformed fullness to the quantization output.

14. The video-coding method of claim **13**, where:

the quantization output comprises:

a minimum quantization parameter; and

a maximum quantization parameter.

15. The video-coding method of claim **13**, where selecting the quantization output comprises mapping the transformed buffer fullness to a first quantization output according to multiple transformed buffer fullness ranges defined for long term quantization parameter control.

16. The video-coding method of claim **15**, further comprising determining a different, second quantization output responsive to the first quantization output and a buffer fullness range defined for defined for short term quantization parameter control.

17. The video-coding method of claim **1**, where:

obtaining the fullness value for the buffer comprises obtaining a fullness value for the buffer that falls outside of a pre-determined valid range of fullness for the buffer; and

applying the transformation to the fullness value to obtain the transformed fullness comprises applying the transformation to the fullness value to obtain a transformed fullness within the pre-determined valid range.

18. The video-coding system of claim **6**, where the quantization adjustment circuitry is configured to allocate a number of target bits for coding the coding units that varies responsive to the amount of prediction information that is available for the coding units by:

allocating a selected number of bits per unit of data to be coded for a first image line; and

allocating fewer than the selected number of bits per unit of data to be coded for a different image line subsequent to the first image line.

19. The video-coding system of claim **6**, where the quantization adjustment circuitry is configured to allocate a number of target bits for coding the coding units that varies responsive to the amount of prediction information that is available for the coding units by: allocating, for a first coding unit additional coding bits than are allocated for a second coding unit with more prediction information than the first coding unit.

20. The video-coding system of claim **6**, further comprising coding circuitry configured to code data for a coding unit of an image by coding a line of an image slice in accord with the number of target bits.

* * * * *